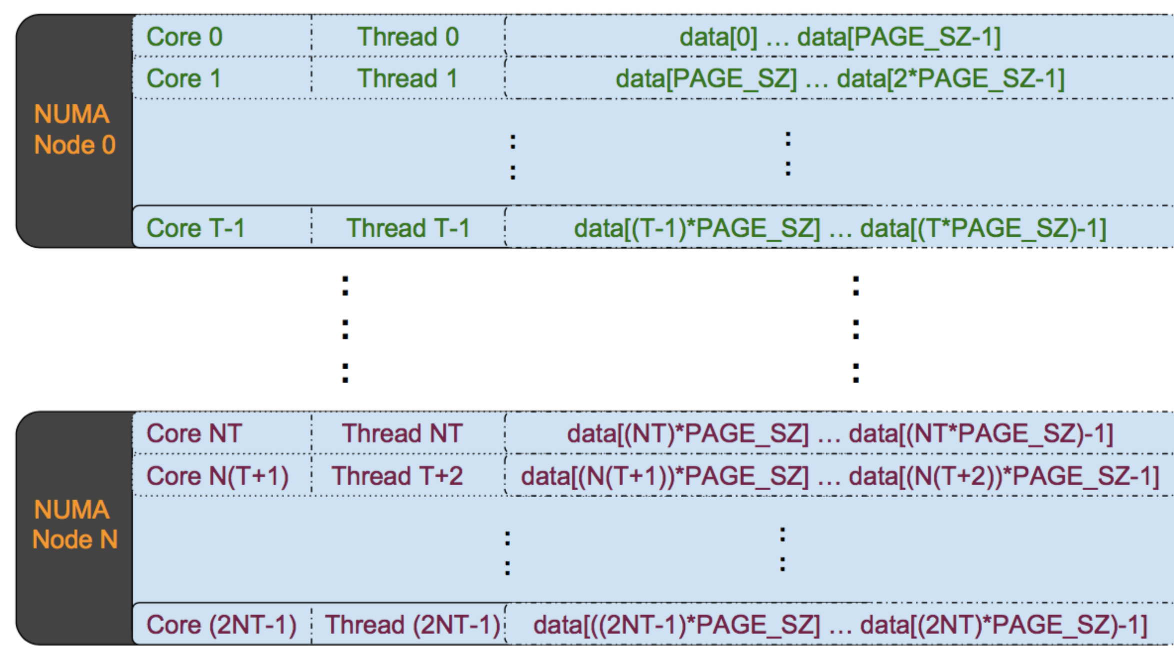
In-memory & Semi-External Memory Large-Scale Clustering

Disa Mhembere

K-means Motivation

- One of the most widely used & important clustering techniques.
- Problems with k-means:
  - Historically does not scale well computation complexity: $O(kdn)$, Space complexity: $O(nd + kd)$
  - Others did not fully utilize CPU resources (toward computation efficiently)
  - Others have not take advantage of NUMA architecture nor new vectorization enabled CPUs
- Our approach:
  - Scale-up not scale-out
- **The how** — Packages:
  - k||means : Shared memory
  - SEM-kmeans, Min-Triangle-SEM-kmeans: Semi-external memory



k||means

- Rethink Lloyd's EM steps. Why not (mostly) merge the two EM-steps into a super-step?
  - How? Share **no** data!
    - Per-thread data structures; combined recursively in || at end of super-step.
  - Result: Embarrassing ||ism
- Developed ||ized initializations via || kmeans++or kmeans|| — they matter

Algorithm 1 k||means algorithm

procedure K||MEANS(V, C, K)
    $pt\vec{Centroids}$ ▷ Per-thread centroids
    $cluster\vec{Assignment}$ ▷ Shared, no conflict
    parfor $\vec{v_i} \in V$ **do**
        for $\vec{c_i} \in C$ **do**
            $[dist_{min}, c\vec{id}_{min}] = min(\mathbf{d}(\vec{v_i}, \vec{c_i}))$
        end for
        $pt\vec{Centroids}[CURR\_THREAD][cid_{min}] \mathrel{+}= \vec{v_i}$
    end parfor
    clusterMeans = mergePtStructs(ptClusters)
end procedure

procedure MERGEPTSTRUCTS($\vec{vectors}$)
    while $|\vec{vectors}| > 1$ **do**
        PAR_MERGE($\vec{vectors}$) ▷ O(T log n)
    end while
    return vectors[0]
end procedure

Old memory bound: $O(nd + kd)$

New memory bound: $O(nd + Tkd)$, *where* $T = \#threads$

k||means Performance on Friendster eigs (66 Mil x 8)

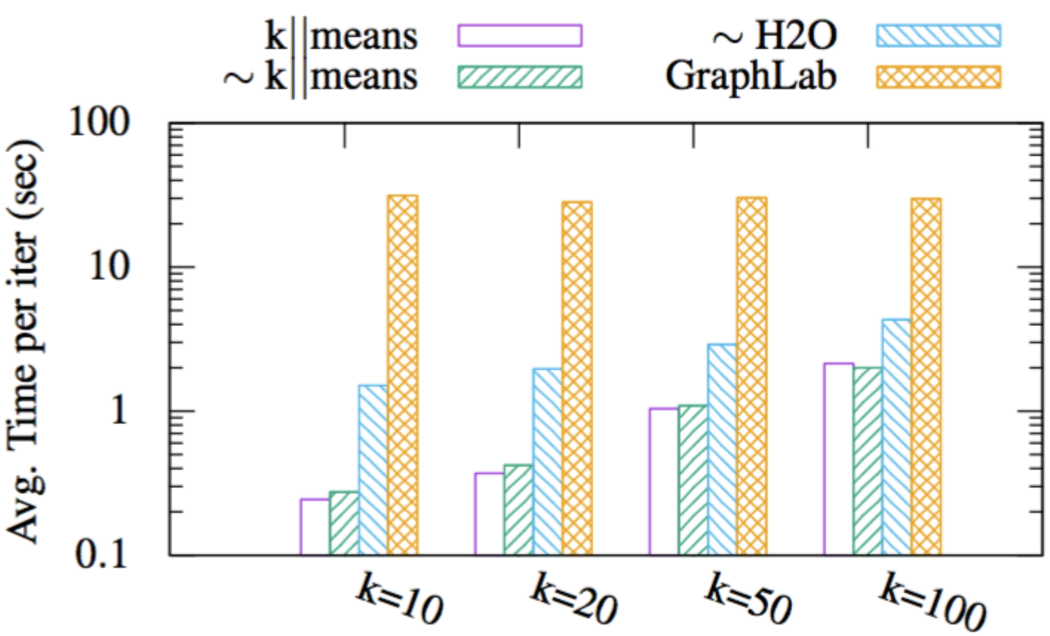


Fig 1: Log scale average time per iteration

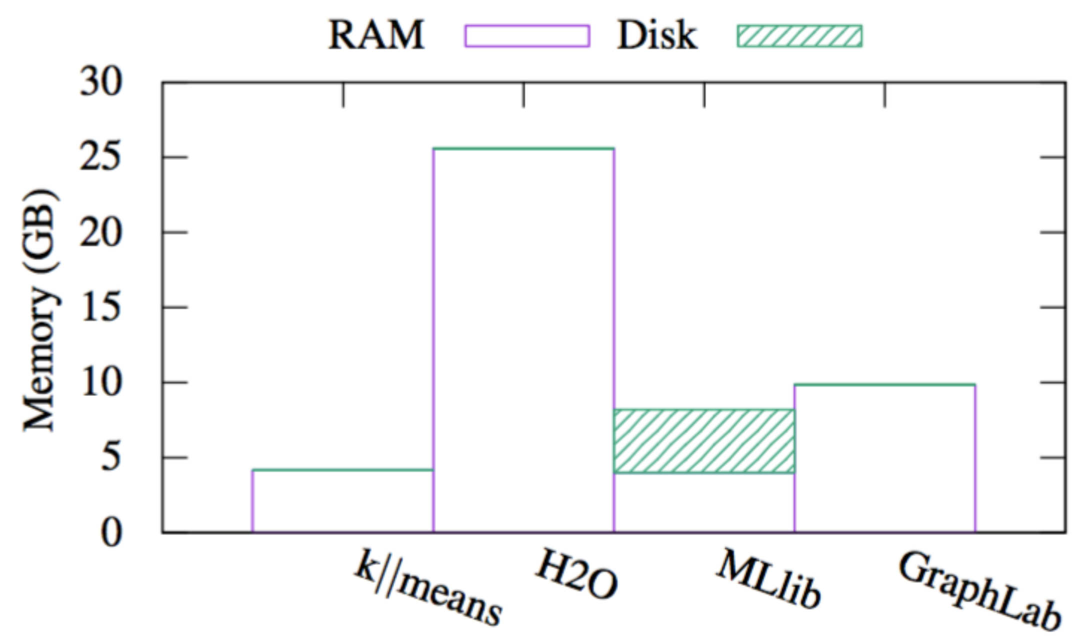


Fig 2: Log scale average time per iteration

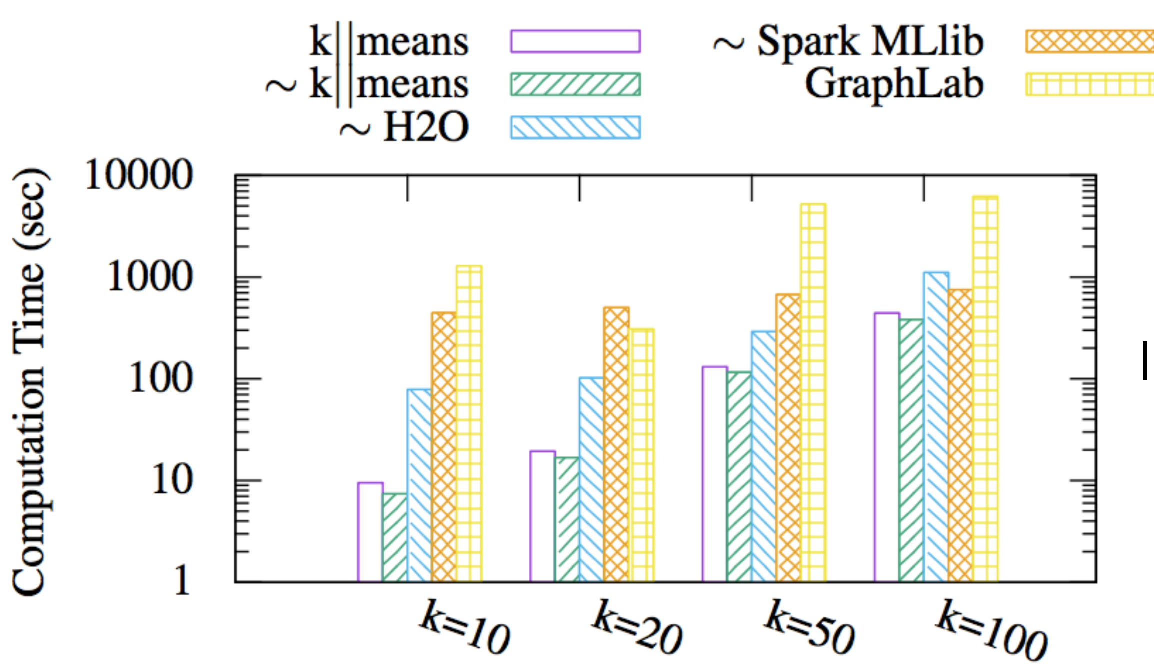


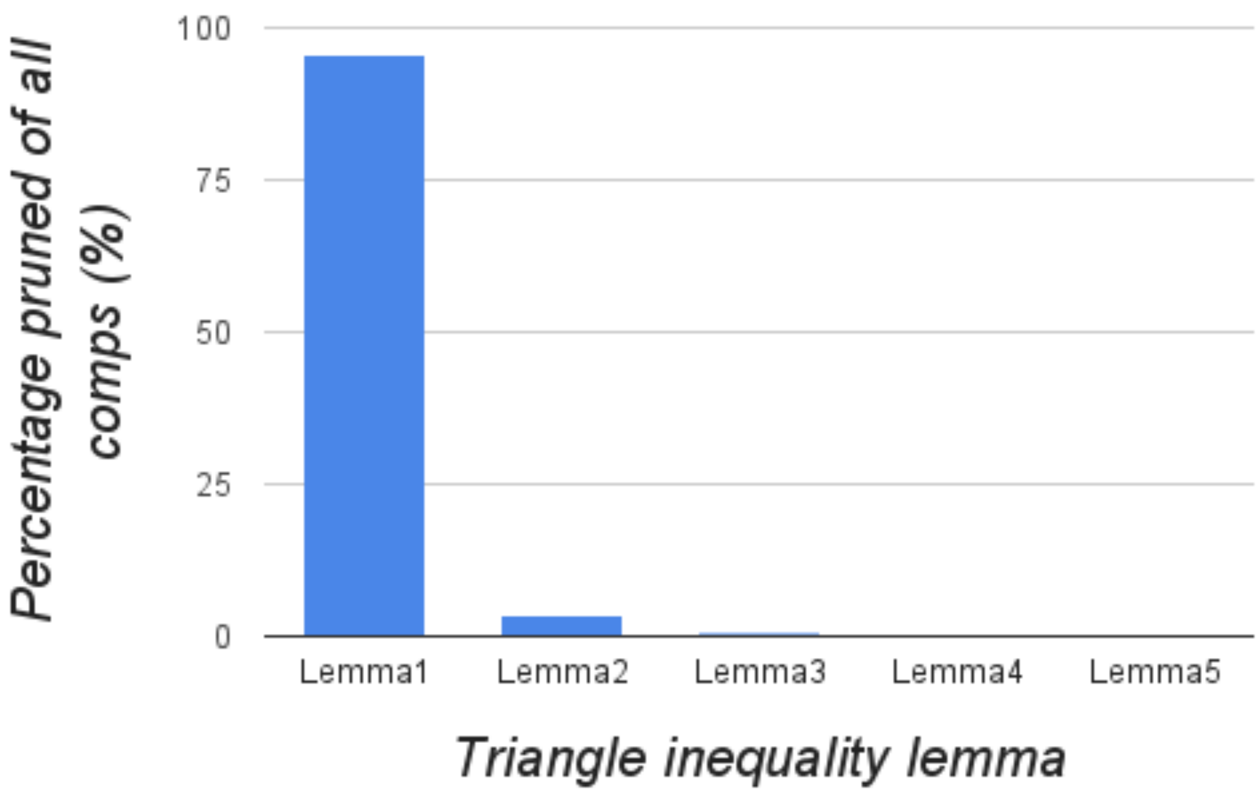
Fig 3: Log scale average computation time. Important because we see **greater** improvement than the per cost per iteration (Fig. 1) due to || initialization modules

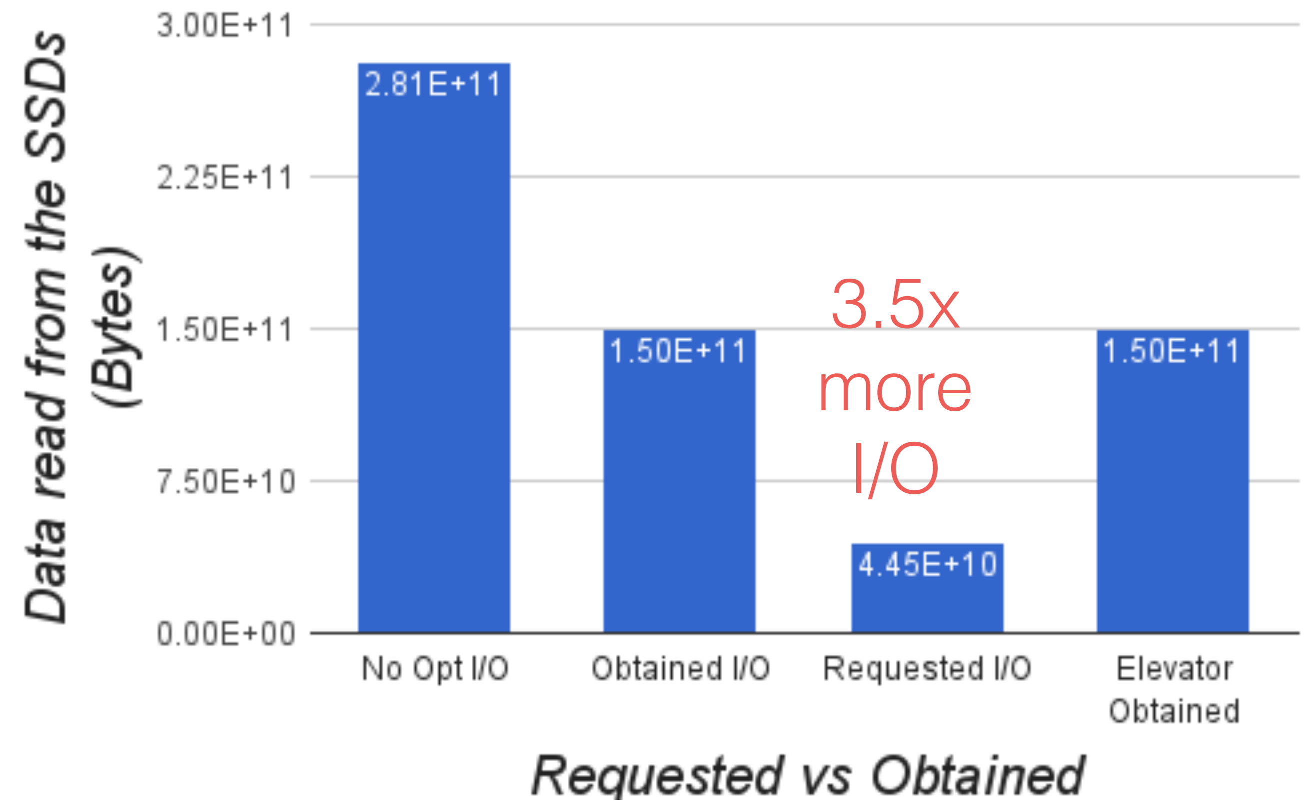
SEM-kmeans

- Can we be comparable with less resources?
  - Semi-external:
    - Memory now $O(n + Tkd) < O(nd + kd) < O(nd + Tkd)$
      - T = #threads/processes and $Tkd << O(nd)$
- Further improve speed using a *modified* triangle inequality pruning algorithm with same memory bound: Min-Triangle-SEM-Kmeans
  - Performance improvement from 2 factors:
    - Reduction in # of distance computations
    - Reduction in I/O complexity

Min-Triangle-SEM-kmeans on Friendster

Micro Data: 66Mil x 8, Dense, Size: 4 GB, K=10



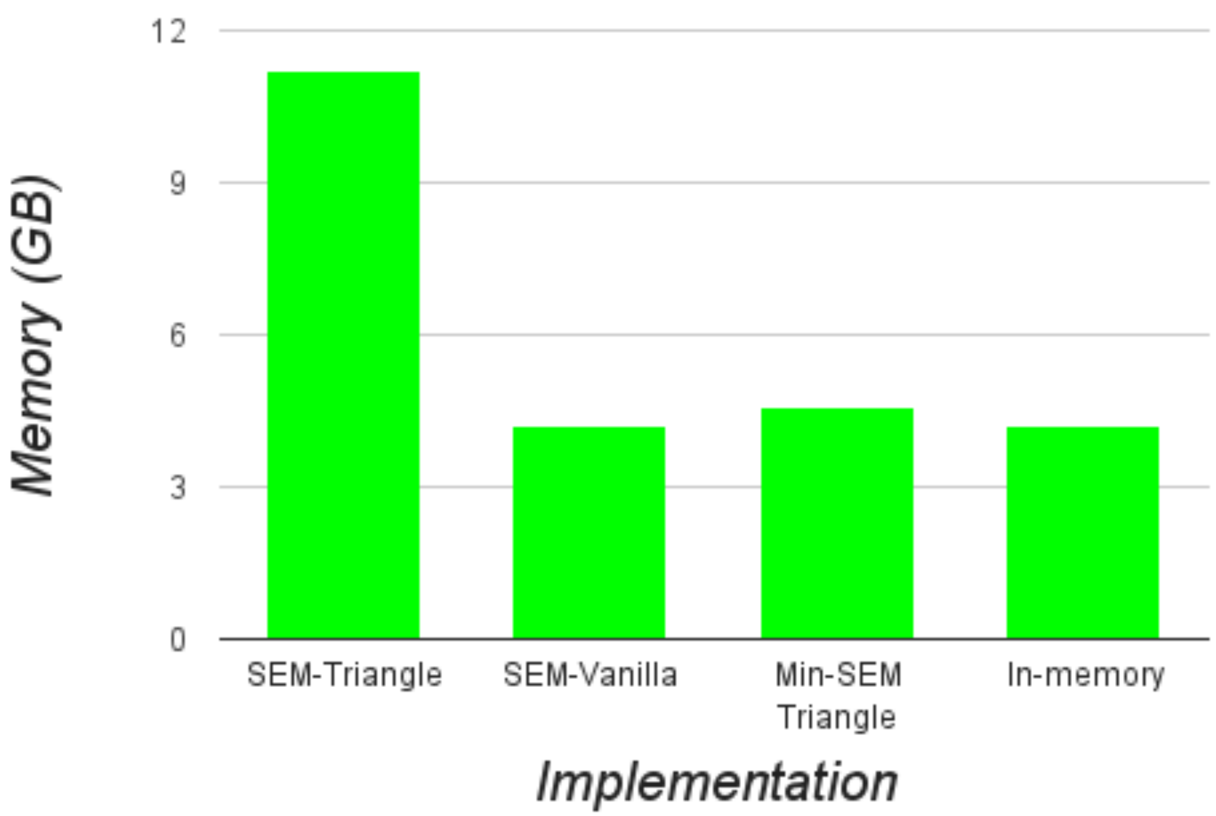


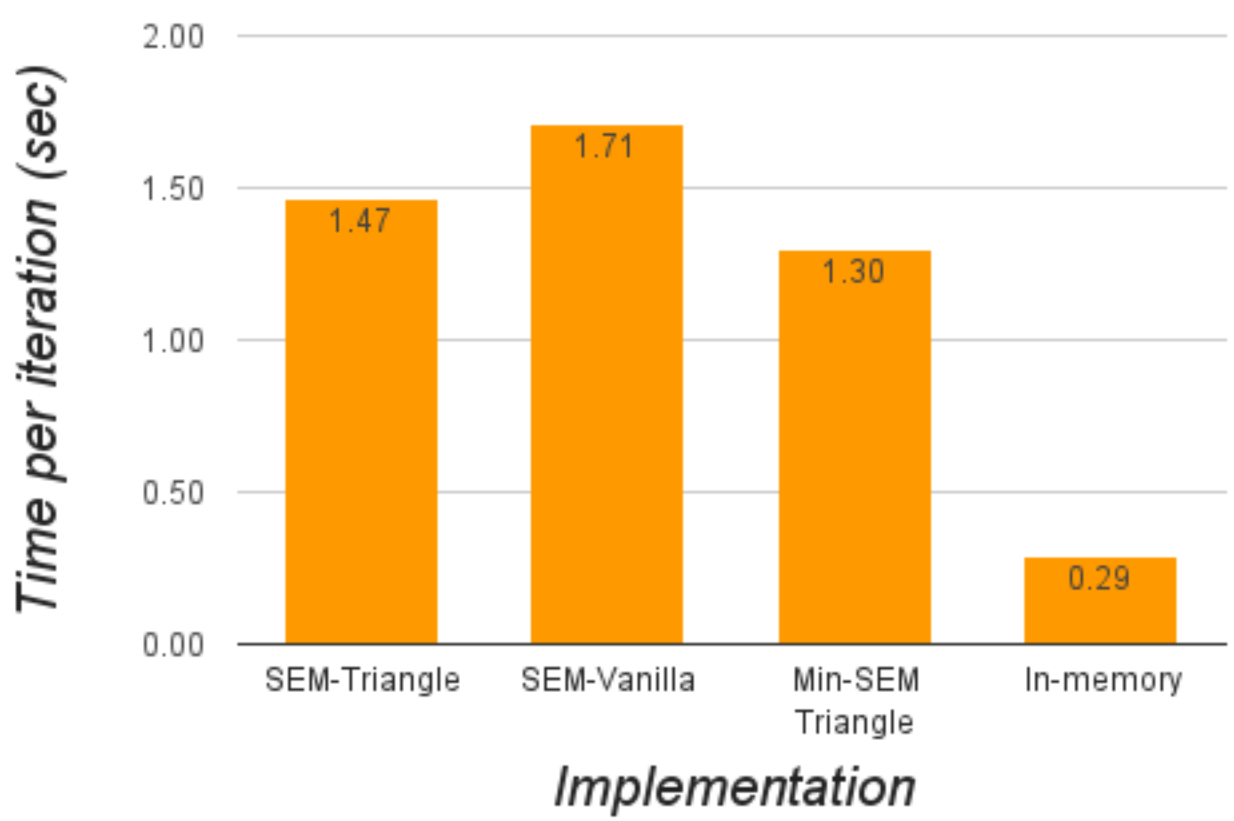
Insight: The effect of upper bound matrix, *O(nk)* is minimal. It contributes only < 2% (Lemma 4 & Lemma 5)

Both depend a lot on the data …
But still very I/O bound

Only **40% CPU** utilization …

Variant comparison on Friendster





*More optimizations to follow,
1GB cache for all SEM

Applications

- Together with the FlashEigen-solver we can perform extremely fast and scalable spectral embedding!
  - Applies to:
    - Connectomics
    - Anomaly detection
    - Machine-Learning
    - NLP
    - A host of other interesting problems:
      - https://sites.google.com/site/dataclusteringalgorithms/clustering-algorithm-applications