Locality Analysis of Graph Reordering Algorithms

Mohsen Koohi Esfahani, Peter Kilpatrick, Hans Vandierendonck
Queen’s University Belfast

Presenter: Zhi Wei Gan

Problem

- Power-Law Distribution of Graphs
  - Leads to Random Memory Accesses
  - Time spent on Memory Accesses = Bottleneck
- Current Graph Reordering Algorithms
  - Improve locality of graph traversals by assigning new IDs to vertices in a way that vertices that are accessed together are read from main memory together
  - Hard to properly measure the performance of these reordering algorithms (excluding pure runtime)
  - Need lightweight metrics and techniques to analyze locality

Definitions

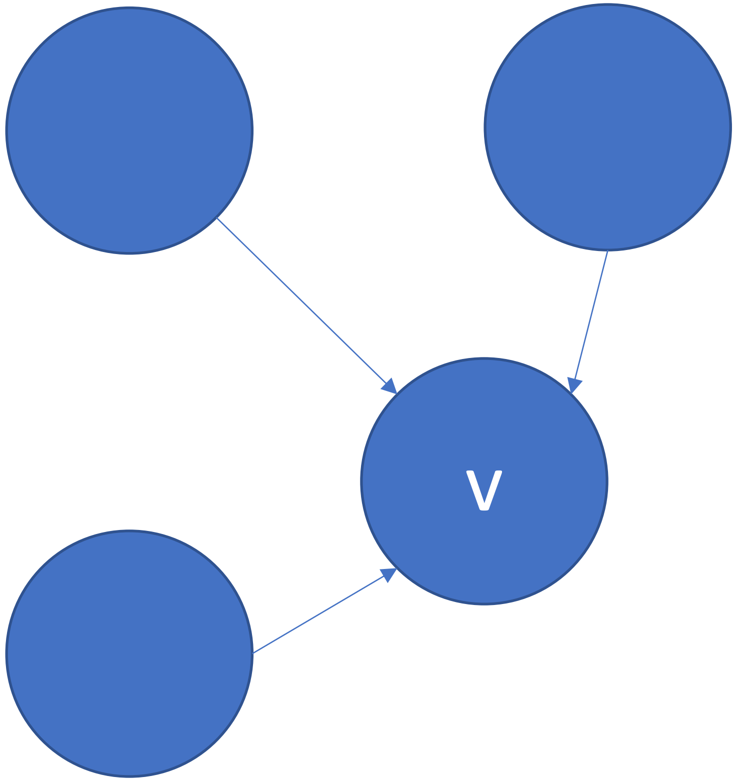
- Low-degree Vertex
  - Less than |E|/|V| edges
- High-degree Vertex
  - More than |E|/|V| edges
- In-hub
  - Vertices with in-degree larger than sqrt(V)
- Out-hub
  - Vertices with out-degree larger than sqrt(V)

Sparse Matrix-Vector (SpMV) multiplication

Algorithm 1: SpMV graph traversal

Input: $G(V, E)$, $\mathbf{D}^i$
Output: $\mathbf{D}^{i+1}$

for $v \in V$ **do**
    $sum = 0;$
    for $u \in v.neighbours$ **do**
        $sum \mathrel{+}= \mathbf{D}^i[u];$
    end
    $\mathbf{D}^{i+1}[v] = sum;$
end



Pull-direction SpMV

Differs from Bucketing/Frontier-based as memory access pattern unpredictable, but can be used as a representative for these types

Datasets

TABLE I: Datasets

Dataset	Name	Source	$\|V\|$ (M)	$\|E\|$ (B)	Type
WebB	WebBase-2001	LWA	115	1.0	WG
TwtrMpi	Twitter MPI	NR	41	1.5	SN
Frndstr	Friendster	NR	65	1.8	SN
SK	SK-Domain	LWA	50	2.0	WG
WbCc	Web-CC12	NR	89	2.0	WG
UKDls	UK-Delis	LWA	110	4.0	WG
UU	UK-Union	LWA	133	5.5	WG
UKDmn	UK-Domain	KN	105	6.6	WG
ClWb9	ClueWeb09	NR	1,700	7.9	WG

SN = Social Network
WG = Web Graph

Locality Types

- Type I: Spatial Reuse, proximity IDs of consecutive neighbors' results in neighbors being placed on the same cache line
- Type II: Temporal Reuse, cache reuses data of some vertex u after using it to process another vertex v.
- Type III: Type II but to a second degree (neighbors of u are also reused)
- Type IV: Reusing a cache line that was used by another thread into a shared cache (Type II but with multithreading)
- Type V: (Type III but with multithreading)

Experimental Setup

- 768 GB Main Memory
- 32KB L1 Cache
- 1MB L2 Cache
- 22MB L3 Shared Cache

Metrics to Measure Locality

- N2N AID (Spatial Locality)
- Cache Miss Rate Degree Distribution (Temporal and Spatio-Temporal Locality)
- Real Execution Performance Metrics:
  - L3 Cache Misses
  - DTLB misses
  - Idle time
  - Effective Cache Size (ECS)

Neighbor to Neighbor Average ID Distance (N2N AID)

- How RAs succeed to bring neighbors close to each other

$$AID_v = \frac{\sum_{i=2}^{i=|N_v|} |N_{v,i} - N_{v,i-1}|}{|N_v|}$$

- Lower AID values = better spatial locality

Cache Miss Rate Degree Distribution

- They collect cache miss rates, but running it on a real machine is time consuming
- They simulated it, but simulating cache miss rates are time consuming for large graphs.
- They optimize their simulations by doing the following:
    - Ignoring execution of non-time-consuming instructions
    - Implemented their own cache replacement policies optimized for SpMV
- Has a 15% error

Graph Reordering Algorithms

- SlashBurn
- Rabbit-Order
- GOrder

SlashBurn (SB)

- Main idea:
  - Finds communities of vertices by removing hubs and finding connected components
  - Assigns consecutive node IDs to hubs of the main graph
- Locality Analysis:
  - Improves locality types IV and V
  - SB is designed for power law graphs, but it only holds true if power-law graphs are deconstructed recursively
    - This doesn’t hold true over different iterations! Reduces locality types I and III
- Real Execution:
  - Destroys spatial locality.

Does SB work?

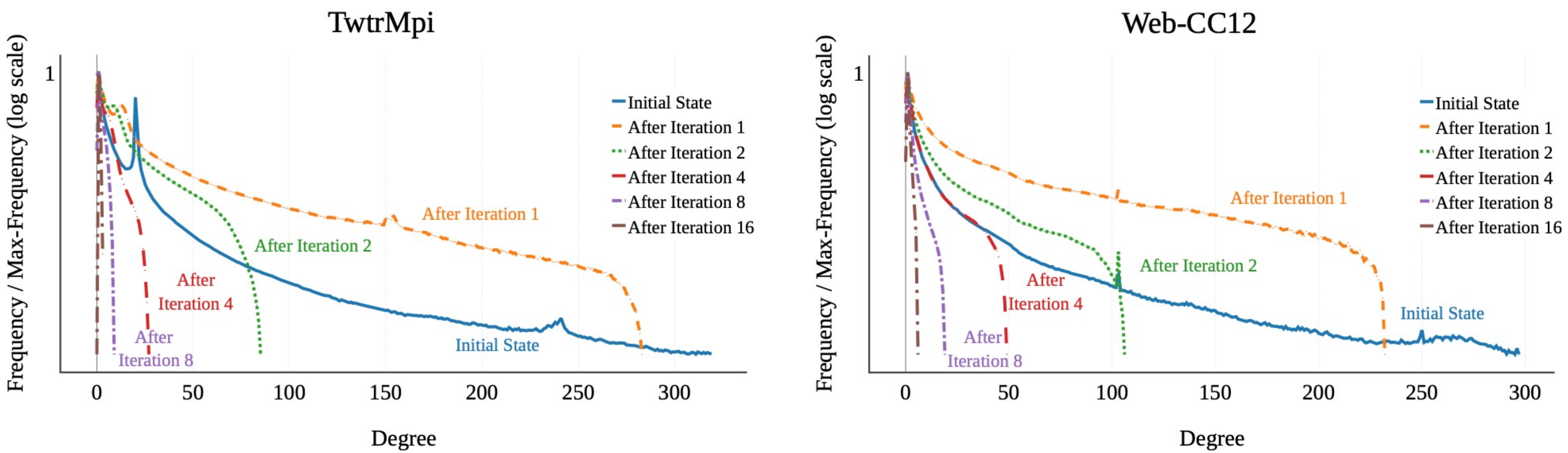


Fig. 2: [Real execution] Degree distribution of initial graph and GCC after SB iterations

Rabbit-Order

- Main idea:
  - Finds communities by using neighbors of vertices.
    - Starts at vertex with lowest degree
    - Searches for neighbor with maximum "gain" that can be reached through merging
    - Merges until there are still "gains" to be made
    - Runs a DFS on the final merged vertices to assign IDs
  - Gain function: $\Delta Q_{u,v} = 2(\frac{w_{uv}}{2|V|} - \frac{deg_u deg_v}{(2|V|)^2})$
- Locality Analysis:
  - Reduces AID of low-degree vertices and improves spatial locality, but the DFS cannot assign consecutive IDs so AID and cache-miss rates are increased for high-degree vertices
- Real Execution:
  - Reduces L3 misses, but execution time is not better.
  - Improving locality does not translate to improved performance since RAs don't change the locality of consecutive vertices, improving locality may increase idle time.

G-Order

- Main idea:
  - Scores between two vertices: $S(u,v) = S_s(u,v) + S_n(u,v)$ where:
    - S_s is the sibling score (the number of common in-neighbors between u and v)
    - S_n is the neighborhood score (the number of edges u and v)
  - Concentrates on temporal reuse instead of identifying communities
- Locality Analysis:
  - Reduces the cache miss rate on high-degree vertices but doesn’t perform well for low-degree vertices
  - Increases the number of reloads of high-degree vertices to provide space in cache for low-degree vertices
- Real execution:
  - Reduces L3 misses

Results

TABLE IV: [Real execution] SpMV execution results (Bl: Baseline without relabeling)

Dataset	**Time** (ms)				**Idle** (%)				**L3 Misses** (M)				**DTLB Misses** (K)			
	Bl	SB	GO	RO	Bl	SB	GO	RO	Bl	SB	GO	RO	Bl	SB	GO	RO
WebB	90	145	89	79	1.5	2.1	2.2	2.3	4.3	6.8	4.3	3.7	0.6	1.7	1.8	1.6
TwtrMpi	354	339	299	366	1.8	2	1.1	1.7	15.7	14.2	12.6	16.3	4.7	2.3	3.1	3.1
Frndstr	771	761	578	667	1.2	1.5	1.4	1.2	40.8	39.2	29.1	34.9	9.3	9.4	7.1	7.6
SK	117	168	109	109	8.2	1.5	1.6	4.1	5.7	8.8	5.5	5.3	0.8	1.4	0.5	0.6
WbCc	438	414	311	297	1.9	2.3	2.3	3.1	20.5	19.3	13.5	12.6	8.6	6.8	6.9	4.5
UKDls	194	317		180	1.9	1.9		2.5	10.1	16.5		9.3	1.8	4.4		1.4
UU	282	486		285	1.9	1.9		6	14.6	24.9		13.8	2.8	7.8		2.4
UKDmn	297	459		281	1.4	2.1		2.7	15.7	23.5		14.7	4.4	5.6		2.7
ClWb9	2,221	2,811			1.3	1.4			100.9	139.3			39M	181		

Locality Analysis of Datasets

- High-degree vertices have close connection to each other in social networks
- Low-degree vertices constitute most web graphs
- Asymmetry: the fraction of in-neighbors that are not out-neighbors for each vertex

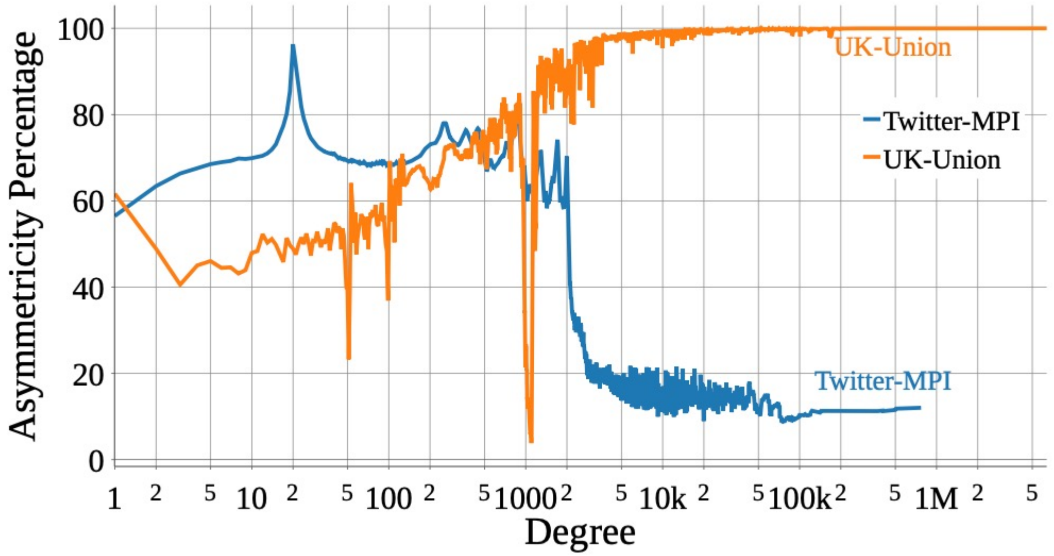
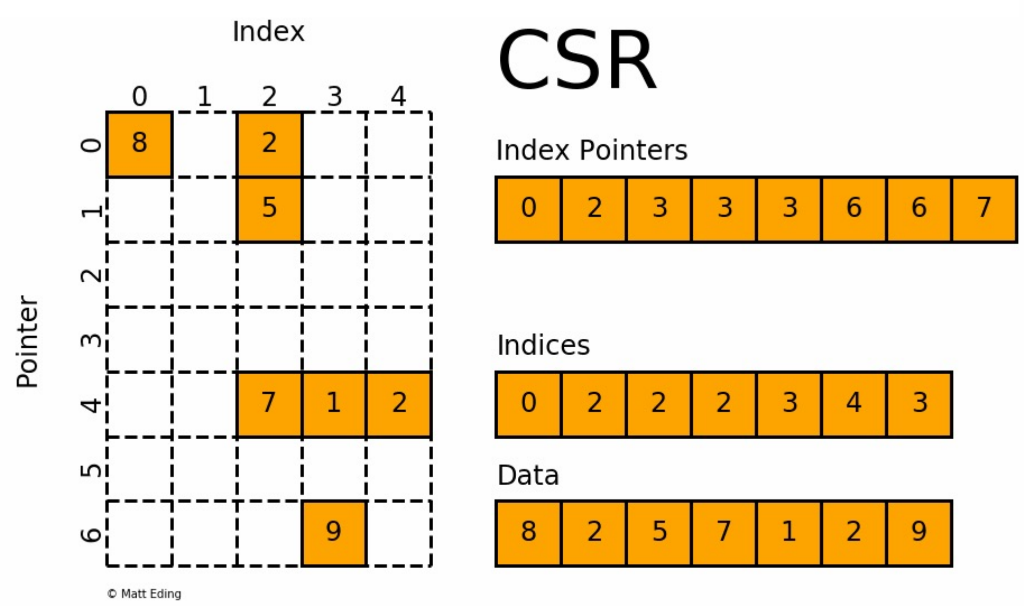
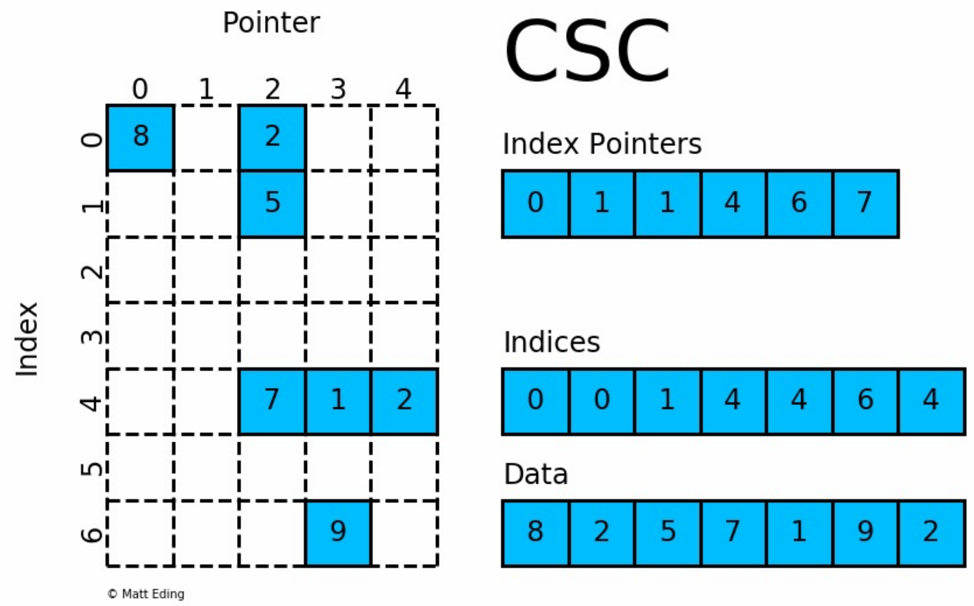


Fig. 4: [Calculation] Asymmetricity degree distribution

CSC vs CSR



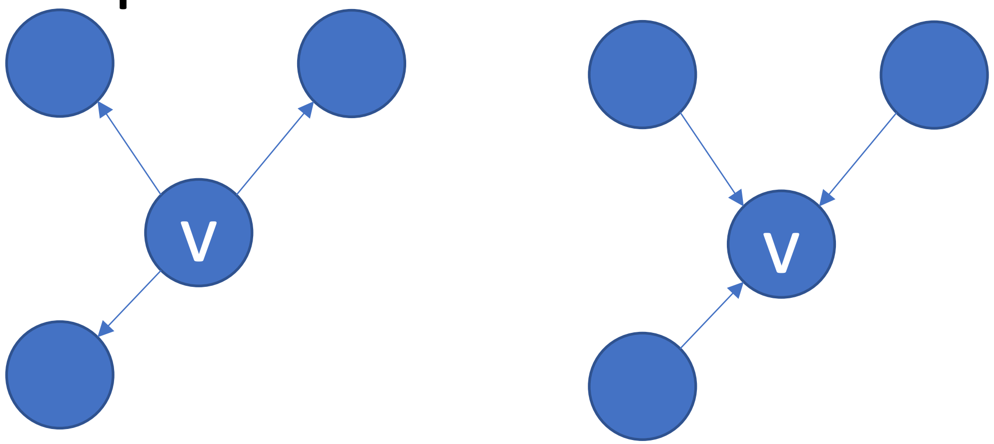
Index
CSR
0 1 2 3 4
0 1 2 3 4 5 6
8 2 5 7 1 2 9
Pointer
Index Pointers
0 2 3 3 3 6 6 7
Indices
0 2 2 2 3 4 3
Data
8 2 5 7 1 2 9
© Matt Eding



Pointer
CSC
0 1 2 3 4
0 1 2 3 4 5 6
8 2 5 7 1 2 9
Index
Index Pointers
0 1 1 4 6 7
Indices
0 0 1 4 4 6 4
Data
8 2 5 7 1 9 2
© Matt Eding

Pull vs. Push Traversal for SpMV

- We use CSR for push
- CSC for pull



push

pull

TABLE VI: [Real execution] CSC vs. CSR read traversals

Dataset	**L3 Misses** (M)		**Traversal Time** (ms)	
	CSC	CSR	CSC	CSR
WebB	4.3	3.8	90	81
TwtrMpi	15,7	21.7	354	439
SK	5.7	4.6	117	88
UKDls	10.1	9.3	194	177
ClWb9	100.9	96.5	2,221	2,129

Why? For pull-traversal, out-hubs have a constructive effect on locality since data is constantly accessed and reused, but for push traversal in-hubs improve locality.

Web graphs are better with CSR traversal
Social networks are better with CSC traversal

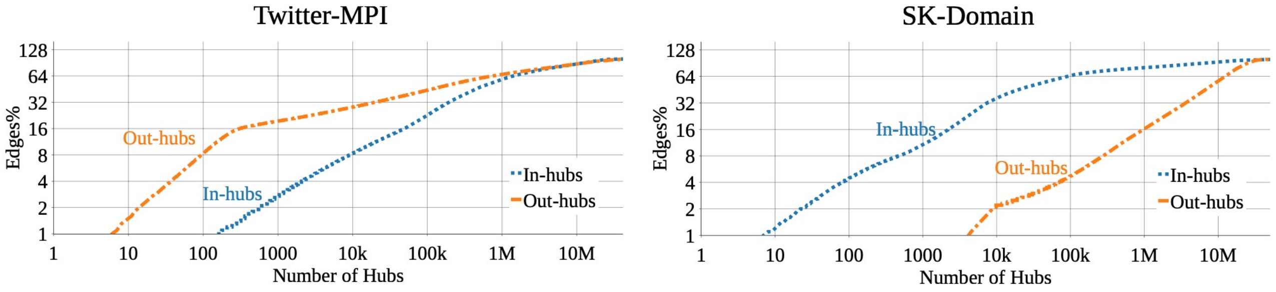


Fig. 6: [Calculation] Comparison of percentage of edges covered by in-hubs in CSR traversal vs. out-hubs in CSC traversal

Optimizing RAs

- SB: continue iterating while GCC-max-degree >= sqrt(V)

Dataset	**Preprocessing** (s)		**Traversal** (ms)		**L3 Misses** (M)	
	SB	SB++	SB	SB++	SB	SB++
TwtrMpi	46	21	339	328	14.2	13.6
Frndstr	75	43	761	700	39.2	36.0
WbCc	81	39	414	334	19.3	14.6

- RO: skip relabeling vertices that are not in an efficacy degree range to reduce preprocessing time and memory