Vertex Reordering for Real-World Graphs and Applications: An Empirical Evaluation

2020 IEEE International Symposium on Workload Characterization

Reet Barik [1], Marco Minutoli [2], Mahantesh Halappanavar [2,1], Nathan R. Tallent [2], Ananth Kalyanaraman [1,2]

[1]School of Electrical Engineering and Computer Science, Washington State University

[2]Pacific Northwest National Laboratory

October 27 – October 29, 2020

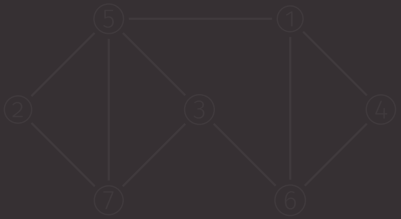
Outline

Vertex Reordering
Motivation and Problem Background

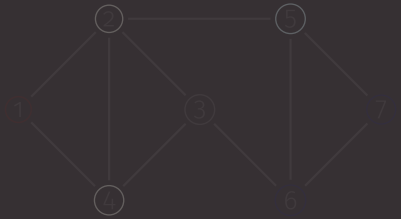
Problem Definition

- Let $G = (V, E)$ denote an input graph, where V is the set of (n) vertices and E is the set of (m) edges.
- We use identifiers in the interval $[1, n]$ to identify the vertices.

A *vertex reordering* Π of V is a 1-1 mapping (bijection or permutation) of V onto a sequence or linear order.



Input Ordering

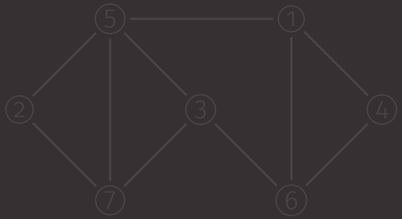


Candidate Ordering

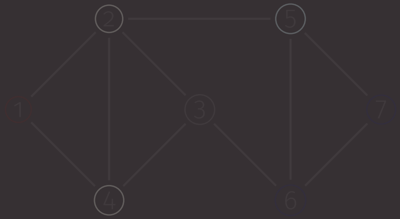
Problem Definition

- Let $G = (V, E)$ denote an input graph, where V is the set of (n) vertices and E is the set of (m) edges.
- We use identifiers in the interval $[1, n]$ to identify the vertices.

A *vertex reordering* Π of V is a 1-1 mapping (bijection or permutation) of V onto a sequence or linear order.



Input Ordering



Candidate Ordering

Problem Definition

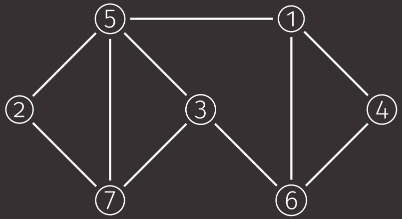
- Let $G = (V, E)$ denote an input graph, where V is the set of (n) vertices and E is the set of (m) edges.
- We use identifiers in the interval $[1, n]$ to identify the vertices.

A *vertex reordering* Π of V is a 1-1 mapping (bijection or permutation) of V onto a sequence or linear order.

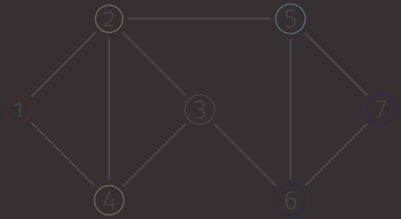
Problem Definition

- Let $G = (V, E)$ denote an input graph, where V is the set of (n) vertices and E is the set of (m) edges.
- We use identifiers in the interval $[1, n]$ to identify the vertices.

A *vertex reordering* Π of V is a 1-1 mapping (bijection or permutation) of V onto a sequence or linear order.



Input Ordering

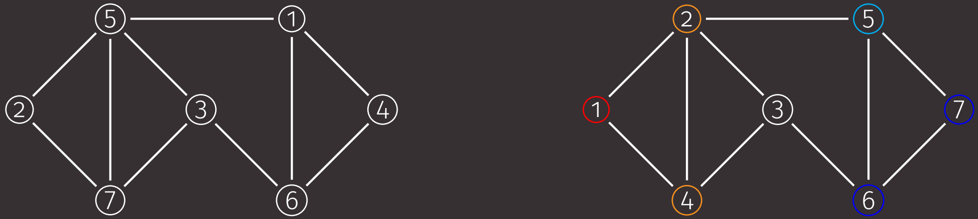


Candidate Ordering

Problem Definition

- Let $G = (V, E)$ denote an input graph, where V is the set of (n) vertices and E is the set of (m) edges.
- We use identifiers in the interval $[1, n]$ to identify the vertices.

A *vertex reordering* Π of V is a 1-1 mapping (bijection or permutation) of V onto a sequence or linear order.



Input Ordering Candidate Ordering

Vertex Reordering
'Gap' Measures

'Gap' Measures

Given an ordering Π of V, the *linear arrangement gap* (or simply *gap*) between any two vertices i and j connected by an edge, is the absolute difference between their ranks in Π.

Gap = | i - j |

- *average gap profile* (or the *average linear arrangement gap*):

$$Avg.Gap(G,\Pi) = \frac{1}{|E|} \sum_{(i,j)\in E} Gap(i,j)$$

A good ordering **minimizes** the Average Linear Gap.

'Gap' Measures

Given an ordering Π of V, the *linear arrangement gap* (or simply *gap*) between any two vertices i and j connected by an edge, is the absolute difference between their ranks in Π.

- *average gap profile* (or the *average linear arrangement gap*):

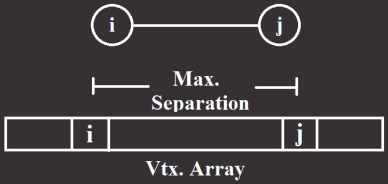
$$Avg.Gap(G, \Pi) = \frac{1}{|E|} \sum_{(i,j) \in E} Gap(i, j)$$

A good ordering **minimizes** the Average Linear Gap.

'Gap' Measures

- *graph bandwidth* (or the *maximum linear arrangement gap*):

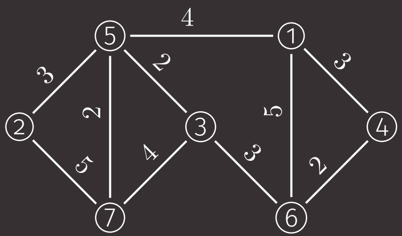
$$Bandwidth(G, \Pi) = \max\{Gap(i,j) | \forall (i,j) \in E\}$$



A good ordering **minimizes** the Graph Bandwidth.

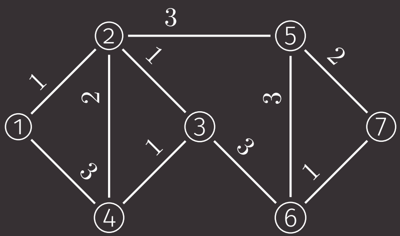
An Example

Natural Ordering



$$Avg.Gap = 3.3$$
$$Bandwidth = 5$$

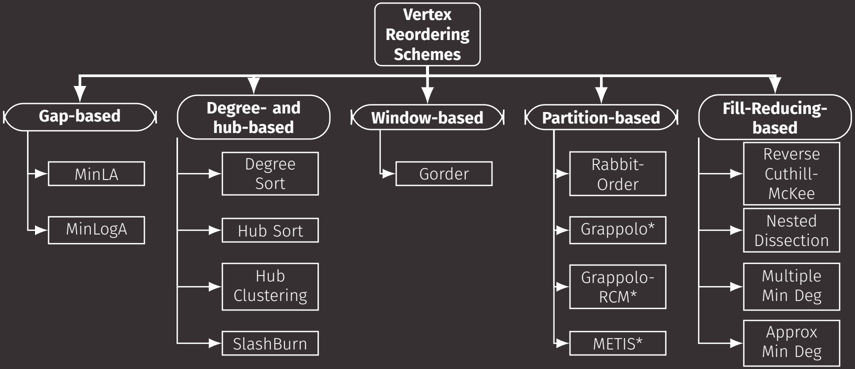
Candidate Ordering



$$Avg.Gap = 1.7$$
$$Bandwidth = 3$$

Vertex Reordering
Classification

Classification of Different Reordering Schemes



Our objective is to compare these reordering schemes from different classes based on their ability to minimize gap-based measures and their impact on real-world graph applications.

Comparative Evaluation
Effect of reordering on Gap Measures

Effect of reordering on Gap Measures

Inputs

Input	#Vertices	#Edges	Δ	Std Dev
Small Instances for Qualitative Analysis				
Chicago Road	1,467	1,298	12	2.539
Euroroad	1,174	1,417	10	1.189
Facebook (NIPS)	2,888	2,981	769	22.888
U. Rovira i Virgili	1,133	5,451	71	9.340
delaunay_n11	2,048	6,128	13	1.392
Figeys	2,239	6,452	314	17.013
US power grid	4,941	6,594	19	1.791
delaunay_n12	4,096	12,265	14	1.367
Hamster small	1,858	12,534	272	20.731
Hamster full	2,426	16,631	273	19.873
PGP	10,680	24,316	205	8.077
delaunay_n13	8,192	24,548	12	1.343
OpenFlights	2,939	30,501	473	43.216
fe_4elt2	11,143	32,819	12	0.890
Twitter lists	23,370	33,101	239	10.143
Google+	23,628	39,242	2,771	35.285
cs4	22,499	43,859	4	0.302
cti	16,840	48,233	6	0.501
delaunay_n14	16,384	49,123	16	1.348
CAIDA	26,475	53,381	2,628	33.374
Vsp	10,498	53,869	229	16.199
wing_nodal	10,937	75,489	28	2.862
Cora citation	23,166	91,500	379	11.314
Gnutella	62,586	147,892	95	5.701
arXiv astro-ph	18,771	198,050	504	30.565

×

Reordering Schemes

- Natural
- METIS
- Degree Sort
- RCM
- Gorder
- Grappolo
- Random
- Grappolo-RCM
- Rabbit-Order
- SlashBurn
- ND

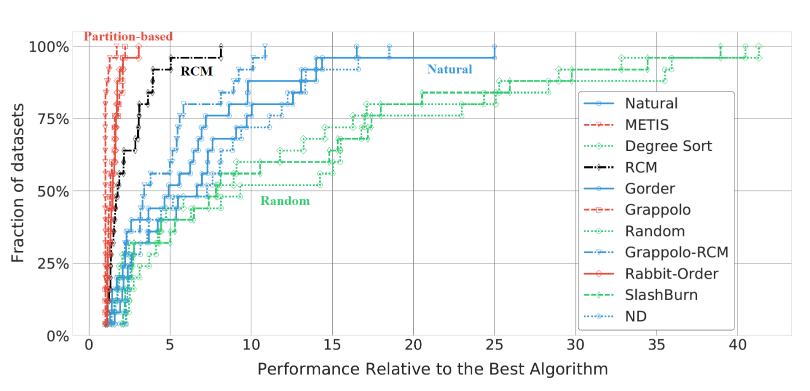
=

Reordering Quality

- Average Linear Gap
- Graph Bandwidth

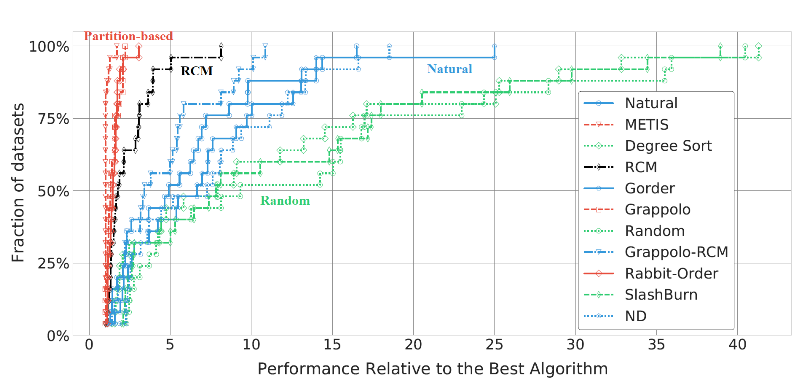
Comparative Evaluation
Results

Average Gap Profile



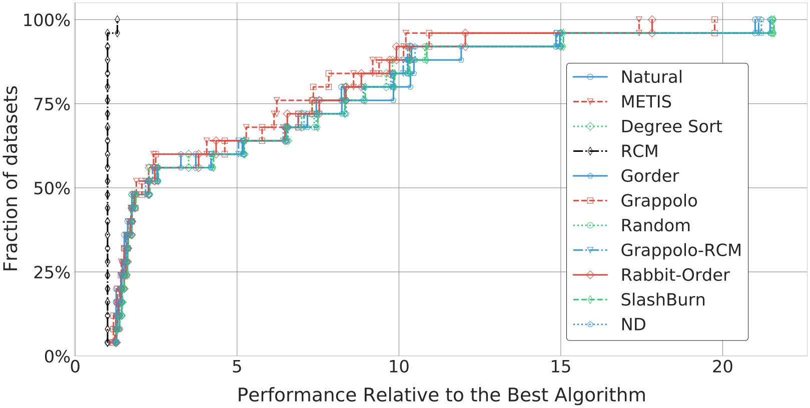
The X-axis represents the factor by which a given scheme fares relative to the best performing scheme over that fraction of inputs.

Average Gap Profile



Takeaway: Partition based schemes (METIS, Grappolo, Rabbit-Order) do better in minimizing the Average Gap Profile.

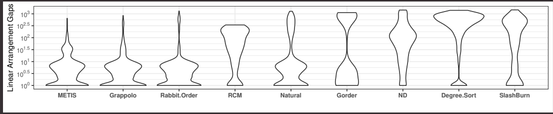
Graph Bandwidth



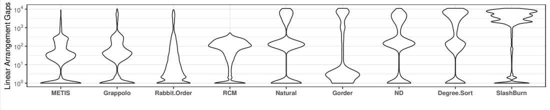
Takeaway: RCM (Reverse Cuthill-McKee) outperforms other schemes in reducing the graph bandwidth.

Comparative Evaluation
Gap Distribution

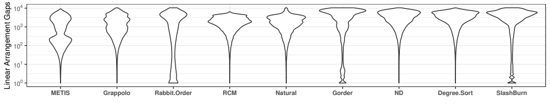
Gap Distribution



$\hat{\xi}$:	12.17	14.04	21.24	99.10	84.45	304.26	202.16	502.44	249.76
β:	648.00	847.00	1,308.00	343.00	1,278.00	1,120.00	1,437.00	1,401.00	1,466.00



$\hat{\xi}$:	124.03	229.59	217.41	103.33	1,447.11	1,350.39	1,379.34	2,591.26	4,023.82
β:	8,988.00	10,193.00	9,196.00	516.00	10,838.00	11,076.00	10,938.00	11,102.00	11,138.00

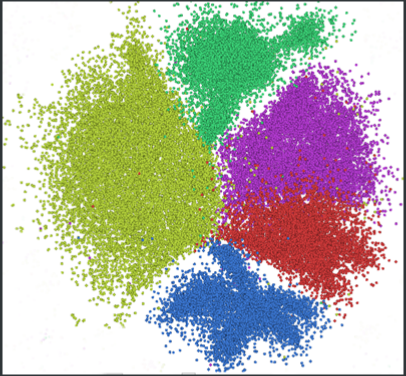


$\hat{\xi}$:	1,392.10	1,964.45	2,323.13	1,876.73	1,986.26	3,872.86	3,528.15	3,439.97	3,786.74
β:	9,015.00	10,369.00	10,457.00	6,405.00	10,386.00	10,456.00	10,433.00	10,288.00	10,495.00

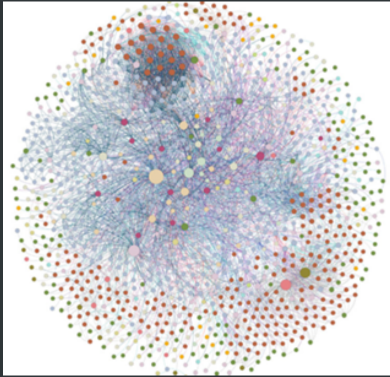
Violin plots of the gap distribution and gap metrics for different methods (green is best, and yellow is worst) for **Chicago** (top-left), **Fe_4elt2** (top-right), and **vsp** (bottom).
A **Bottom-Heavy Distribution** indicates **good reordering scheme**.

Real-World Application Study

Real-World Application Study



Community Detection



Influence Maximization

Real-World Application Study

Effect of reordering on performance

Effect of reordering on performance

Inputs

Input	#Vertices	#Edges	Δ	Std Dev
Large Instances for Application Performance Analysis				
Livemocha	1.04E+05	2.19E+06	2,980	1.10E+02
California Roadnet	1.97E+06	2.77E+06	12	9.95E-01
Hyves	1.40E+06	2.78E+06	31,883	4.53E+01
arXiv hep-ph	2.81E+04	4.60E+06	11,134	5.91E+02
Youtube	3.22E+06	9.38E+06	91,751	1.28E+02
Skitter	1.70E+06	1.11E+07	35,455	1.37E+02
Actor collaborations	3.82E+05	3.31E+07	16,764	4.22E+02
LiveJournal links	5.20E+06	4.87E+07	15,016	5.06E+01
Orkut	3.07E+06	1.17E+08	33,313	1.55E+02

×

Reordering Schemes

- Natural
- METIS
- Degree Sort
- RCM
- Grappolo

=

Impact on

- Community Detection: Grappolo
- Influence Maximization: Ripples

Test Platform

Shared memory server with 8 Intel Xeon Platinum 8276 (Cascade Lake) CPUs & 6 TB DDR4-2933 of memory (L1 cache of 32KB; per-core L2 cache of 1 MB, & (socket-wide) L3 cache of 38.5 MB)

Real-World Application Study
Community Detection Results

Impact on Community Detection

	Phase (s)				Iteration (s)				Iteration Count				Modularity (final)				Work% (CPU Time)				Work/edge (loads)			
Graph	Grappolo	RCM	Natural	Degree	Grplo	RCM	Ntrl	Degr	Grplo	RCM	Ntrl	Degr	Grplo	RCM	Ntrl	Degr	Grplo	RCM	Ntrl	Degr	Grplo	RCM	Ntrl	Degr
LiveMocha	1.4	1.0	1.5	1.5	0.20	0.26	0.37	0.51	7	4	4	3	0.040	0.047	0.027	0.019	28%	24%	20%	19%	3.1	3.2	3.8	2.8
CA RoadNet	0.9	1.5	0.9	1.9	0.13	0.38	0.23	0.65	7	4	4	3	0.992	0.992	0.992	0.992	9%	12%	9%	16%	0.5	1.5	0.8	2.5
Hyves	1.6	1.5	1.7	2.4	0.39	0.39	0.44	0.60	4	4	4	4	0.608	0.731	0.722	0.715	22%	19%	18%	17%	2.5	2.5	2.1	2.1
arXiv hep-ph	3.3	3.8	4.2	5.8	0.09	0.10	0.12	0.14	36	37	36	42	0.516	0.530	0.531	0.535	50%	45%	39%	36%	0.9	1.0	0.9	0.9
YouTube	4.2	3.4	9.5	7.3	0.28	0.85	0.48	1.82	15	4	20	4	0.644	0.636	0.644	0.633	21%	12%	19%	13%	6.4	7.1	6.0	7.7
Skitter	6.3	2.9	8.1	5.8	0.14	0.95	0.21	1.45	44	3	39	4	0.842	0.833	0.840	0.827	20%	11%	15%	12%	3.2	7.0	3.4	7.3
Actor collab	7.7	12.4	8.5	29.2	0.16	0.26	0.21	0.58	49	48	41	50	0.708	0.715	0.714	0.717	32%	24%	27%	20%	2.3	2.3	2.4	2.5
LiveJournal	24.0	49.5	52.3	90.6	0.25	0.92	0.66	1.41	96	54	79	64	0.746	0.751	0.749	0.741	35%	29%	27%	30%	9.9	10.9	10.4	11.7
Orkut	70.1	131.7	52.9	131.1	0.55	1.25	0.62	2.11	128	105	85	62	0.623	0.622	0.635	0.623	38%	44%	38%	41%	8.8	10.9	9.4	11.9
	lower (redder) better				*lower (redder) better*				*lower (redder) better*				*higher (redder) better*				*higher (redder) better*				*lower (redder) better*			

Takeaways:

- **Phase and Iteration times**: Grappolo usually outperforms Degree Sort, at times by factors $2\times - 4\times$ or more.
- **Parallel Efficiency (Work%)**: The Grappolo ordering usually has the highest It also has the lowest work per edge. This tends to result in a better load balance.

Impact on Community Detection

Graph	Phase (s)				Iteration (s)				Iteration Count				Modularity (final)				Work% (CPU Time)				Work/edge (loads)			
	Grappolo	RCM	Natural	Degree	Grplo	RCM	Ntrl	Degr	Grplo	RCM	Ntrl	Degr	Grplo	RCM	Ntrl	Degr	Grplo	RCM	Ntrl	Degr	Grplo	RCM	Ntrl	Degr
LiveMocha	1.4	1.0	1.5	1.5	0.20	0.26	0.37	0.51	7	4	4	3	0.040	0.047	0.027	0.019	28%	24%	20%	19%	3.1	3.2	3.8	2.8
CA RoadNet	0.9	1.5	0.9	1.9	0.13	0.38	0.23	0.65	7	4	4	3	0.992	0.992	0.992	0.992	9%	12%	9%	16%	0.5	1.5	0.8	2.5
Hyves	1.6	1.5	1.7	2.4	0.39	0.39	0.44	0.60	4	4	4	4	0.608	0.731	0.722	0.715	22%	19%	18%	17%	2.5	2.5	2.1	2.1
arXiv hep-ph	3.3	3.8	4.2	5.8	0.09	0.10	0.12	0.14	36	37	36	42	0.516	0.530	0.531	0.535	50%	45%	39%	36%	0.9	1.0	0.9	0.9
YouTube	4.2	3.4	9.5	7.3	0.28	0.85	0.48	1.82	15	4	20	4	0.644	0.636	0.644	0.633	21%	12%	19%	13%	6.4	7.1	6.0	7.7
Skitter	6.3	2.9	8.1	5.8	0.14	0.95	0.21	1.45	44	3	39	4	0.842	0.833	0.840	0.827	20%	11%	15%	12%	3.2	7.0	3.4	7.3
Actor collab	7.7	12.4	8.5	29.2	0.16	0.26	0.21	0.58	49	48	41	50	0.708	0.715	0.714	0.717	32%	24%	27%	20%	2.3	2.3	2.4	2.5
LiveJournal	24.0	49.5	52.3	90.6	0.25	0.92	0.66	1.41	96	54	79	64	0.746	0.751	0.749	0.741	35%	29%	27%	30%	9.9	10.9	10.4	11.7
Orkut	70.1	131.7	52.9	131.1	0.55	1.25	0.62	2.11	128	105	85	62	0.623	0.622	0.635	0.623	38%	44%	38%	41%	8.8	10.9	9.4	11.9
	lower (redder) better				*lower (redder) better*				*lower (redder) better*				*higher (redder) better*				*higher (redder) better*				*lower (redder) better*			

Takeaways:

- **Phase and Iteration times**: Grappolo usually outperforms Degree Sort, at times by factors $2\times - 4\times$ or more.
- **Parallel Efficiency (Work%)**: The Grappolo ordering usually has the highest It also has the lowest work per edge. This tends to result in a better load balance.

Impact on Community Detection

Graph	Phase (s) Grappolo	Phase (s) RCM	Phase (s) Natural	Phase (s) Degree	Iteration (s) Grplo	Iteration (s) RCM	Iteration (s) Ntrl	Iteration (s) Degr	Iteration Count Grplo	Iteration Count RCM	Iteration Count Ntrl	Iteration Count Degr	Modularity (final) Grplo	Modularity (final) RCM	Modularity (final) Ntrl	Modularity (final) Degr	Work% (CPU Time) Grplo	Work% (CPU Time) RCM	Work% (CPU Time) Ntrl	Work% (CPU Time) Degr	Work/edge (loads) Grplo	Work/edge (loads) RCM	Work/edge (loads) Ntrl	Work/edge (loads) Degr
LiveMocha	1.4	1.0	1.5	1.5	0.20	0.26	0.37	0.51	7	4	4	3	0.040	0.047	0.027	0.019	28%	24%	20%	19%	3.1	3.2	3.8	2.8
CA RoadNet	0.9	1.5	0.9	1.9	0.13	0.38	0.23	0.65	7	4	4	3	0.992	0.992	0.992	0.992	9%	12%	9%	16%	0.5	1.5	0.8	2.5
Hyves	1.6	1.5	1.7	2.4	0.39	0.39	0.44	0.60	4	4	4	4	0.608	0.731	0.722	0.715	22%	19%	18%	17%	2.5	2.5	2.1	2.1
arXiv hep-ph	3.3	3.8	4.2	5.8	0.09	0.10	0.12	0.14	36	37	36	42	0.516	0.530	0.531	0.535	50%	45%	39%	36%	0.9	1.0	0.9	0.9
YouTube	4.2	3.4	9.5	7.3	0.28	0.85	0.48	1.82	15	4	20	4	0.644	0.636	0.644	0.633	21%	12%	19%	13%	6.4	7.1	6.0	7.7
Skitter	6.3	2.9	8.1	5.8	0.14	0.95	0.21	1.45	44	3	39	4	0.842	0.833	0.840	0.827	20%	11%	15%	12%	3.2	7.0	3.4	7.3
Actor collab	7.7	12.4	8.5	29.2	0.16	0.26	0.21	0.58	49	48	41	50	0.708	0.715	0.714	0.717	32%	24%	27%	20%	2.3	2.3	2.4	2.5
LiveJournal	24.0	49.5	52.3	90.6	0.25	0.92	0.66	1.41	96	54	79	64	0.746	0.751	0.749	0.741	35%	29%	27%	30%	9.9	10.9	10.4	11.7
Orkut	70.1	131.7	52.9	131.1	0.55	1.25	0.62	2.11	128	105	85	62	0.623	0.622	0.635	0.623	38%	44%	38%	41%	8.8	10.9	9.4	11.9
	lower (redder) better				*lower (redder) better*				*lower (redder) better*				*higher (redder) better*				*higher (redder) better*				*lower (redder) better*			

Takeaways:

- **Phase and Iteration times**: Grappolo usually outperforms Degree Sort, at times by factors $2\times - 4\times$ or more.
- **Parallel Efficiency (Work%)**: The Grappolo ordering usually has the highest It also has the lowest work per edge. This tends to result in a better load balance.

Impact on Community Detection

	YouTube					Skitter					Actor collab					LiveJournal					Orkut				
Order	Lat	L1	L2	L3	DRAM	Lat	L1	L2	L3	DRAM	Lat	L1	L2	L3	DRAM	Lat	L1	L2	L3	DRAM	Lat	L1	L2	L3	DRAM
Grappolo	13	9%	16%	26%	65%	9	14%	21%	15%	53%	11	13%	49%	15%	16%	19	5%	11%	15%	31%	14	8%	14%	21%	8%
RCM	34	14%	14%	26%	62%	8	25%	11%	22%	39%	12	13%	48%	17%	15%	21	8%	8%	19%	29%	23	10%	11%	26%	15%
Natural	11	10%	7%	19%	78%	11	13%	21%	12%	52%	9	15%	49%	11%	16%	25	7%	8%	14%	37%	16	8%	13%	20%	14%
Degree	16	14%	13%	18%	60%	13	18%	12%	28%	34%	13	10%	51%	18%	11%	31	8%	7%	19%	35%	20	10%	12%	24%	14%

Expectation:

- lower memory latency to correspond to memory boundedness at lower memory levels.
- lower iteration time to correlate to lower memory latency.

Reality:

- Neither holds in all cases. For example: Grappolo tends to be more DRAM bound than Degree, even though average memory latency is lower

Impact on Community Detection

	YouTube					Skitter					Actor collab					LiveJournal					Orkut				
Order	Lat	L1	L2	L3	DRAM	Lat	L1	L2	L3	DRAM	Lat	L1	L2	L3	DRAM	Lat	L1	L2	L3	DRAM	Lat	L1	L2	L3	DRAM
Grappolo	13	9%	16%	26%	65%	9	14%	21%	15%	53%	11	13%	49%	15%	16%	19	5%	11%	15%	31%	14	8%	14%	21%	8%
RCM	34	14%	14%	26%	62%	8	25%	11%	22%	39%	12	13%	48%	17%	15%	21	8%	8%	19%	29%	23	10%	11%	26%	15%
Natural	11	10%	7%	19%	78%	11	13%	21%	12%	52%	9	15%	49%	11%	16%	25	7%	8%	14%	37%	16	8%	13%	20%	14%
Degree	16	14%	13%	18%	60%	13	18%	12%	28%	34%	13	10%	51%	18%	11%	31	8%	7%	19%	35%	20	10%	12%	24%	14%

Expectation:

- lower memory latency to correspond to memory boundedness at lower memory levels.
- lower iteration time to correlate to lower memory latency.

Reality:

- Neither holds in all cases. For example: Grappolo tends to be more DRAM bound than Degree, even though average memory latency is lower

Impact on Community Detection

	YouTube					Skitter					Actor collab					LiveJournal					Orkut				
Order	Lat	L1	L2	L3	DRAM	Lat	L1	L2	L3	DRAM	Lat	L1	L2	L3	DRAM	Lat	L1	L2	L3	DRAM	Lat	L1	L2	L3	DRAM
Grappolo	13	9%	16%	26%	65%	9	14%	21%	15%	53%	11	13%	49%	15%	16%	19	5%	11%	15%	31%	14	8%	14%	21%	8%
RCM	34	14%	14%	26%	62%	8	25%	11%	22%	39%	12	13%	48%	17%	15%	21	8%	8%	19%	29%	23	10%	11%	26%	15%
Natural	11	10%	7%	19%	78%	11	13%	21%	12%	52%	9	15%	49%	11%	16%	25	7%	8%	14%	37%	16	8%	13%	20%	14%
Degree	16	14%	13%	18%	60%	13	18%	12%	28%	34%	13	10%	51%	18%	11%	31	8%	7%	19%	35%	20	10%	12%	24%	14%

Expectation:

- lower memory latency to correspond to memory boundedness at lower memory levels.
- lower iteration time to correlate to lower memory latency.

Reality:

- Neither holds in all cases. For example: Grappolo tends to be more DRAM bound than Degree, even though average memory latency is lower

Impact on Community Detection

	YouTube					Skitter				
Order	Lat	L1	L2	L3	DRAM	Lat	L1	L2	L3	DRAM
Grappolo	13	9%	16%	26%	65%	9	14%	21%	15%	53%
RCM	34	14%	14%	26%	62%	8	25%	11%	22%	39%
Natural	11	10%	7%	19%	78%	11	13%	21%	12%	52%
Degree	16	14%	13%	18%	60%	13	18%	12%	28%	34%

	Actor collab					LiveJournal					Orkut				
Order	Lat	L1	L2	L3	DRAM	Lat	L1	L2	L3	DRAM	Lat	L1	L2	L3	DRAM
Grappolo	11	13%	49%	15%	16%	19	5%	11%	15%	31%	14	8%	14%	21%	8%
RCM	12	13%	48%	17%	15%	21	8%	8%	19%	29%	23	10%	11%	26%	15%
Natural	9	15%	49%	11%	16%	25	7%	8%	14%	37%	16	8%	13%	20%	14%
Degree	13	10%	51%	18%	11%	31	8%	7%	19%	35%	20	10%	12%	24%	14%

Expectation:

- lower memory latency to correspond to memory boundedness at lower memory levels.
- lower iteration time to correlate to lower memory latency.

Reality:

- Neither holds in all cases. For example: Grappolo tends to be more DRAM bound than Degree, even though average memory latency is lower

Possible Explanation

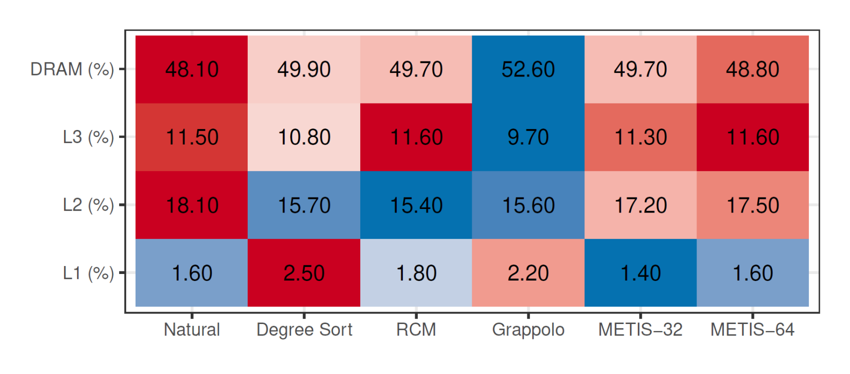
- Graph traversal costs **may not** be the dominant fraction of an algorithm's execution time.
- An algorithm's use of auxiliary data structures can result in additional memory access patterns that negate the benefits of vertex orderings in graph traversals.

Possible Explanation

- Graph traversal costs **may not** be the dominant fraction of an algorithm’s execution time.
- An algorithm’s use of auxiliary data structures can result in additional memory access patterns that negate the benefits of vertex orderings in graph traversals.

Real-World Application Study
Influence Maximization Results

Impact on Influence Maximization



Memory performance counters for the hotspot function in Ripples for the input graph 'Skitter'. We report how often the machine was stalled at all the layers of the memory hierarchy (L1/L2/L3/DRAM).

Takeaways

Expectation:

- Reordering schemes should shift the runtime profile to be more cache bound rather than memory bound and a consequent performance improvement.

Reality:

- Overall improvements on Ripples are marginal, with no particular reordering scheme standing out.

Takeaways

Expectation:

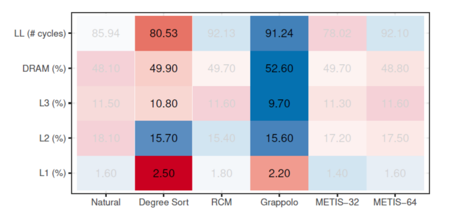
- Reordering schemes should shift the runtime profile to be more cache bound rather than memory bound and a consequent performance improvement.

Reality:

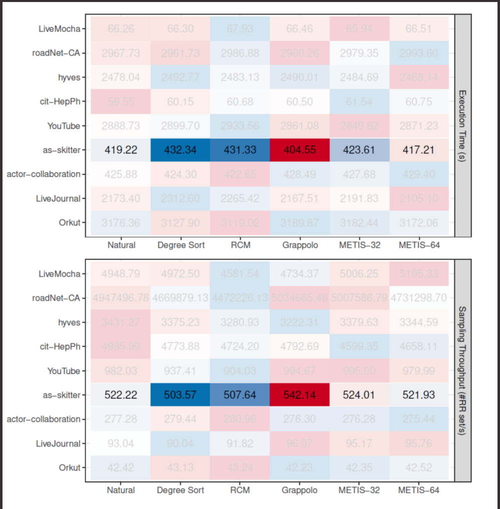
- Overall improvements on Ripples are marginal, with no particular reordering scheme standing out.

Example

Degree Sort and Grappolo based orderings show a significant improvement on the percentage of memory operation bound by the L1 cache.



Example



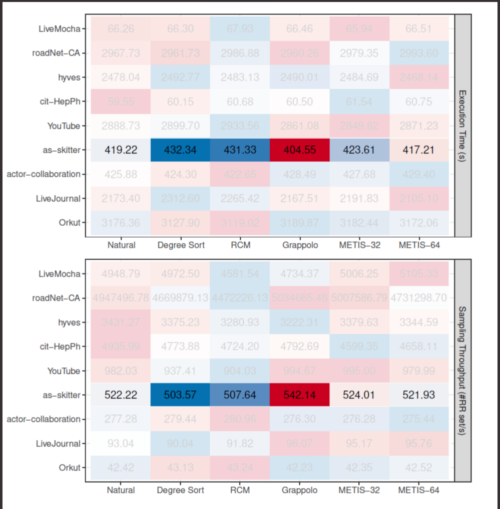
Expectation:

- Observing corresponding good performance (Sampling Throughput and Execution Time) for both Degree Sort and Grappolo ordering.

Reality:

- Degree Sort and Grappolo are at the opposite of the execution time and sampling throughput spectrum.

Example



Expectation:

- Observing corresponding good performance (Sampling Throughput and Execution Time) for both Degree Sort and Grappolo ordering.

Reality:

- Degree Sort and Grappolo are at the opposite of the execution time and sampling throughput spectrum.

Possible Explanation

- Parallel threads competing for memory bandwidth and cache space.

Solution

- If the underlying implementation can be made locality-aware such applications can also benefit from ordering schemes.

Possible Explanation

- Parallel threads competing for memory bandwidth and cache space.

Solution

- If the underlying implementation can be made locality-aware such applications can also benefit from ordering schemes.

Conclusions

- Partition based reordering schemes generally tend to do better than others in optimizing for Average Gap profiles.
- RCM does best in terms of reducing the Graph Bandwidth.
- Reordering has more benefits when it comes to Community Detection as an end application than Influence Maximization.

Conclusions

- Partition based reordering schemes generally tend to do better than others in optimizing for Average Gap profiles.
- RCM does best in terms of reducing the Graph Bandwidth.
- Reordering has more benefits when it comes to Community Detection as an end application than Influence Maximization.

Conclusions

- Partition based reordering schemes generally tend to do better than others in optimizing for Average Gap profiles.
- RCM does best in terms of reducing the Graph Bandwidth.
- Reordering has more benefits when it comes to Community Detection as an end application than Influence Maximization.

Conclusions

Does reordering matter? **YES!!**

- Idea about memory hierarchy usage by the end application.

Is reordering worth it? Depends...

- Cost amortization by repeated use of the input graph in graph analytic applications.

Conclusions

Does reordering matter? **YES!!**

- Idea about memory hierarchy usage by the end application.

Is reordering worth it? Depends...

- Cost amortization by repeated use of the input graph in graph analytic applications.

Conclusions

Does reordering matter? **YES!!**

- Idea about memory hierarchy usage by the end application.

Is reordering worth it? Depends...

- Cost amortization by repeated use of the input graph in graph analytic applications.

Conclusions

Does reordering matter? **YES!!**

- Idea about memory hierarchy usage by the end application.

Is reordering worth it? Depends...

- Cost amortization by repeated use of the input graph in graph analytic applications.

Conclusions

Does reordering matter? **YES!!**

- Idea about memory hierarchy usage by the end application.

Is reordering worth it? Depends...

- Cost amortization by repeated use of the input graph in graph analytic applications.

Conclusions

Does reordering matter? **YES!!**

- Idea about memory hierarchy usage by the end application.

Is reordering worth it? Depends...

- Cost amortization by repeated use of the input graph in graph analytic applications.

Acknowledgments

This research is in parts supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration, through the ExaGraph project at the Pacific Northwest National Laboratory (PNNL); by the U.S. National Science Foundation (NSF) grants CCF 1815467, OAC 1910213, and CCF 1919122 to Washington State University. PNNL is operated by Battelle Memorial Institute under Contract DE-AC06-76RL01830.

Thank You