USC University of Southern California

Accelerating PageRank using Partition-Centric Processing

Kartik Lakhotia, Rajgopal Kannan, Viktor Prasanna

USENIX ATC'18

Outline

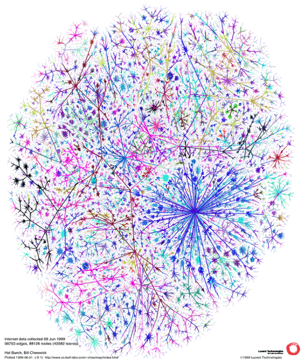
- **Introduction**
- Partition-centric Processing Methodology
- Analytical Evaluation
- Experimental Results
- Generalization
- Conclusion

Graph Analytics

- Graphs → ubiquitously preferred data representation

Social network

Internet



Road Network

…

- Era of Big Data, Era of large Graphs
  - Billions of nodes and edges
  - Need high performance processing

PageRank

- Fundamental Node Ranking algorithm
  - Iteratively compute weighted sum of neighbor's $PR[]$

$$PR_{i+1}(v) = \frac{1-d}{|V|} + d \sum_{u \in N_i(v)} \frac{PR_i(u)}{|N_o(u)|}$$

- Important benchmark for the performance of
  - Graph Analytics
  - Sparse Matrix Vector multiplication
    - core kernel of many scientific and engg. applications

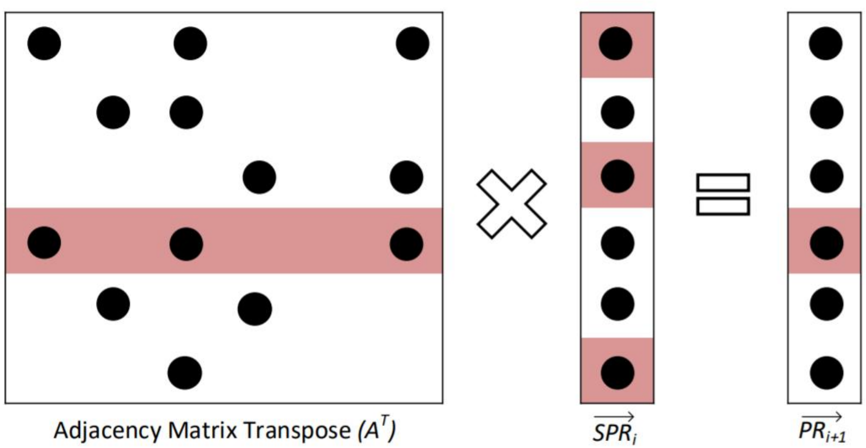
Challenges: Pull Direction PageRank (PDPR)

for $v \in V$ **do**
  $temp = 0$
  for all $u \in N_i(v)$ **do**
    $temp += PR[u]$
  $PR_{next}[v] = \frac{(1-d)\times|V|^{-1} + d\times temp}{|N_o(v)|}$
$\mathrm{swap}(PR, PR_{next})$

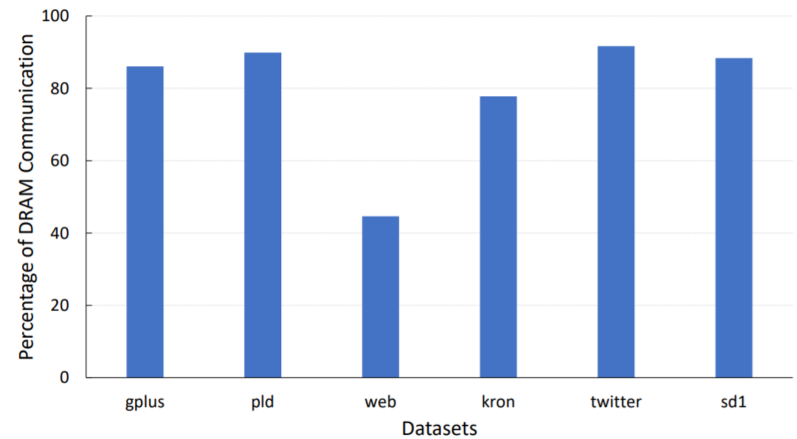
1. PDPR Algorithm

Read $PR[u]$ → fine-grained random memory accesses

- ↓ cache line utilization, ↑ DRAM traffic
- ↓ sustained memory bandwidth
- Cache misses, CPU stalls



2. Random accesses to $\overrightarrow{SPR}$



3. DRAM traffic due to random accesses

Challenges: Vertex-Centric GAS (BVGAS)

- State-of-the-art method[1,2]
  - $Scatter \rightarrow \forall u \in V$, write $msg = \{PR[u], v\} \; \forall \; v \in N_o(u)$ *(semi-sorted on v)*
  - $Gather \rightarrow$ Read msg and accumulate $PR[u]$ into $PR[v]$
  - ↑ cache line utilization; prevent CPU stalls

- Drawbacks:
  - Traverse *entire* graph twice
    - inherently sub-optimal
  - oblivious to vertex ordering induced locality
  - coarse-grained random accesses → poor DRAM BW

1. Buono, Daniele, et al. "Optimizing sparse matrix-vector multiplication for large-scale data analytics." *Proceedings of the 2016 International Conference on Supercomputing*. ACM, 2016
2. Beamer, Scott, et al. "Reducing PageRank communication via propagation blocking." *Proceedings of Parallel and Distributed Processing Symposium*. IEEE, 2017

Contributions

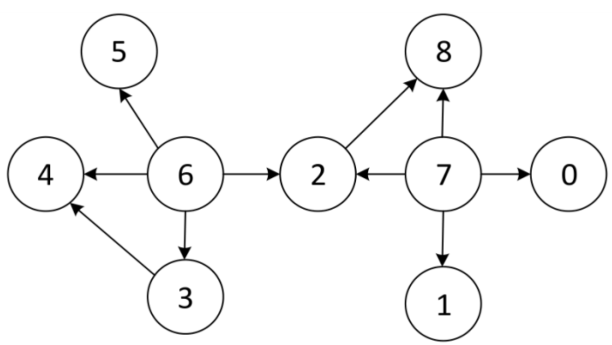
- Novel **Partition-centric Processing Methodology**
  - enables efficient Processor-DRAM communication
- Optimizations to address communication challenges
  - Partition-centric update propagation → ↓ DRAM traffic
  - Partition-Node Graph Data Layout → **sequential** DRAM accesses
  - Branch avoidance mechanism → remove data-dependent branches
- Achieves
  - upto **4.7 GTEPS** sustained throughput using 16 cores
  - upto **77%** of peak DRAM bandwidth
- Applicable to weighted graphs and generic SpMV computation

Outline

- Introduction
- **Partition-centric Processing Methodology**
- Analytical Evaluation
- Experimental Results
- Generalization
- Conclusion

Graph Partitioning

- $Partitions$ → disjoint $cacheable$ sets of vertices
- Partition-centric abstraction of $Graph$ → set of links between nodes and partitions
  - unlocks comm. efficiency not achievable with VC/EC paradigms
- Index based partitioning
  - simple, low pre-processing overhead



Partitions

P_0	P_1	P_2
0	3	6
1	4	7
2	5	8

Example graph with partitions of size 3

Partition-Centric Processing Methodology (PCPM)

- Partition-Centric Processing with GAS model
  - $Scatter$ messages to neighbouring partitions
  - $Gather$ incoming messages to compute new PageRank values
- Write messages in statically allocated disjoint memory spaces ($bins$)
  - no locks/atomics, ↑ scalability
  - $Dest.ID$ written only in first iteration, ↓ comm.
- Each thread processes 1 partition at a time
  - Vertex data $cacheable$
  - low latency random access

Updates	Dest. ID
PR[6]	2
PR[7]	0
	1
	2

Bin 0

Updates	Dest. ID
PR[3]	4
PR[6]	3
	4
	5

Bin 1

Updates	Dest. ID
PR[2]	8
PR[7]	8

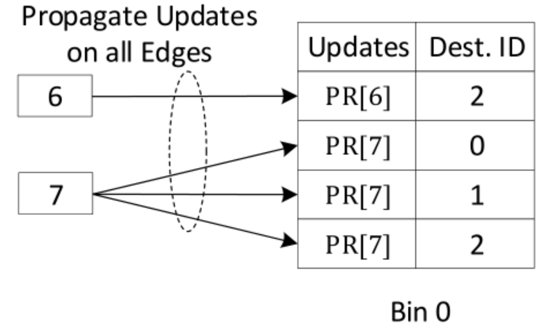
Bin 2

Optimization 1: Partition-Centric Update Propagation

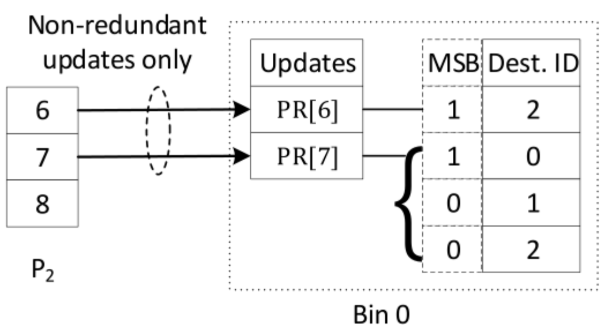
- $Single$ update from a node to all neighbours in a partition
  - Natural outcome of PC abstraction
  - Drastically reduce communication volume

- MSB of destination IDs for demarcation
  - read new update if MSB = 1

- Issues to address
  - *Scatter*
    - traverses *unused* edges $\{(7,1),(7,2)\}$
    - switch bins for each update insertion
  - *Gather*
    - Data-dependent unpredictable branches due to MSB check



(a) Scatter in Vertex-centric GAS



(b) Scatter in PCPM

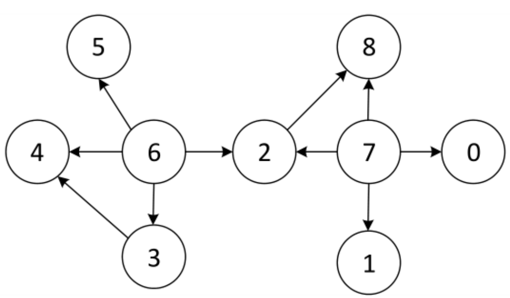
Optimization 2: Data Layout

- Bipartite Partition-Node Graph (PNG)
  - at most 1 edge between node and partition
  - eliminate *unused* edge traversal
- Group the edges by destination partition
  - All updates to one bin at a time
  - Random access to vertices
- Create PNG on a per-partition basis
  - Vertices cached, DRAM accesses *sequential*

Algorithm PCPM scatter phase using PNG layout

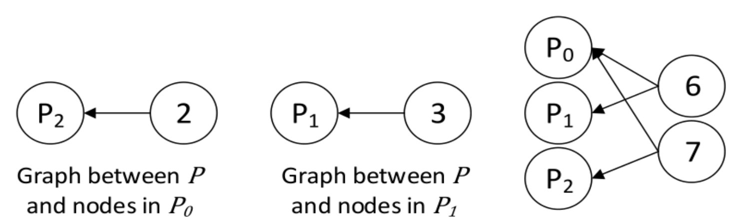
$G'(P,V,E') \rightarrow$ PNG, $N_i^p(p') \rightarrow$ in-neighbors of partition p' in bipartite graph of partition p

for all $p \in P$ **do in parallel** ▷ **Scatter**
  for all $p' \in P$ **do**
    for all $u \in N_i^p(p')$ **do**
      insert $PR[u]$ into $update\_bins[p']$



Partitions

P_0	P_1	P_2
0	3	6
1	4	7
2	5	8



Graph between P and nodes in P_0

Graph between P and nodes in P_1

Graph between P and nodes in P_2

1. Original Graph

2. PNG Layout

Optimization 3: Branch Avoidance

- $Gather$ uses pointers to read bins
  - $destID\_ptr$ for $destID\_bins$
  - $update\_ptr$ for $update\_bins$

- When to increment pointers?
  - $destID\_ptr$ every iteration
  - $update\_ptr$ if MSB = 1

- Directly add MSB to $update\_ptr$
  - no branch based cond. check on MSB

Algorithm Branch Avoiding gather function in PCPM

```
PR[:] = 0
for all p ∈ P do in parallel                         ▷ Gather
    {destID_ptr, update_ptr} ← 0
    while destID_ptr < size(destID_bins[p]) do
        id ← destID_bins[p][destID_ptr++]
        update_ptr += MSB(id)
        id ← id & bitmask
        PR[id] += update_bins[p][update_ptr]
```

Outline

- Introduction
- Partition-centric Processing Methodology
- **Analytical Evaluation**
- Experimental Results
- Generalization
- Conclusion

Parameters

- Original Graph $G(V, E)$
  - $n = |V|$
  - $m = |E|$
- PNG Layout $G'(P, V, E')$
  - $E' \rightarrow$ edges between nodes and partitions
  - $k = |P| =$ # partitions
  - $^{*}r = \frac{|E|}{|E'|} \geq 1$
- Software
  - $d_v =$ sizeof (updates) $= 4B/8B$
  - $d_i =$ sizeof (index) $= 4B$
- Cache
  - $^{*}c_{mr} =$ PDPR cache miss ratio
  - $l =$ sizeof (cache line) $= 64B$

* r and c_{mr} are a function of graph locality. As locality increases, $r \uparrow$ and $c_{mr} \downarrow$

DRAM Communication Model

Method	Communication Volume
$PDPR_{comm}$	$m(d_i + c_{mr} l)$
$BVGAS_{comm}$	$2m(d_i + d_v)$
$PCPM_{comm}$	$m\left(d_i\left(1 + \frac{1}{r}\right) + \frac{2d_v}{r}\right)$

- $BVGAS_{comm}$ oblivious to locality
  - good if locality is low and c_{mr} is high
- $PCPM_{comm} \leq BVGAS_{comm}$
  - good if locality is low and c_{mr} is high
  - linear in $\frac{1}{r}$ → good for high locality graphs as well

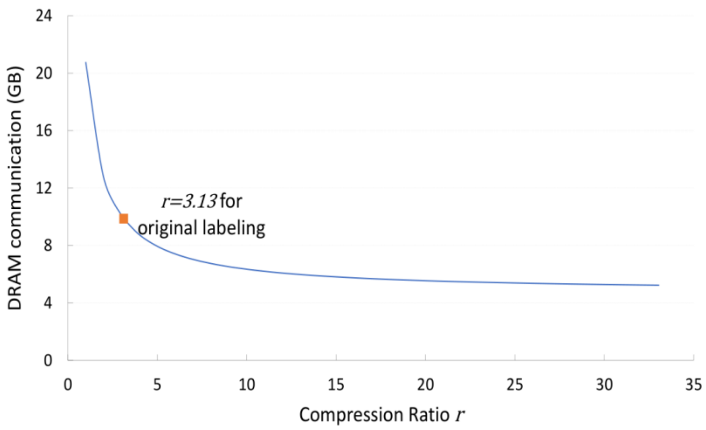


Figure : Predicted DRAM traffic for *kron* graph with $n = 33.5$ M, $m = 1070$ M, $k = 512$ and $d_i = d_v = 4$ Bytes.

Random Access Model

Method	# Random DRAM accesses
$PDPR_{ra}$	mc_{mr}
* $BVGAS_{ra}$	$\frac{md_v}{l}$
$PCPM_{ra}$	k^2

- $PCPM_{ra} \ll BVGAS_{ra} < PDPR_{ra}$
- Example → $kron$ graph
  - $n = 33.5M, m = 1.05B, k = 512, l = 64B$
  - $PCPM_{ra} \approx 0.26M \ll BVGAS_{ra} \approx 67M$

*Assuming full cache line utilization for BVGAS

Outline

- Introduction
- Partition-centric Processing Methodology
- Analytical Evaluation
- **Experimental Results**
- Generalization
- Conclusion

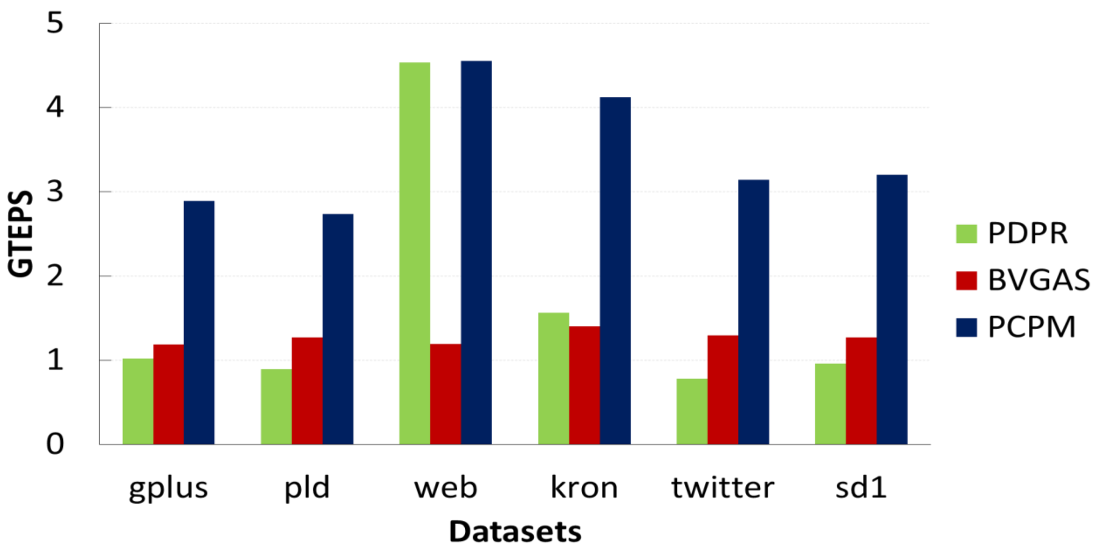
Experimental Setup

- Large real-life and synthetic graphs

Datasest	Description	# Nodes (M)	# Edges (M)
gplus	Google+ social network	28.9	463
pld	Pay-level-domain (web crawl)	42.9	623
web	Webbase-2001 (high locality)	118.1	992.8
Kron	Synthetic (high density)	33.5	1048
twitter	Follower network	61.6	1468.4
sd1	Subdomain graph (web crawl)	95	1937.5

- Intel Xeon E5-2650 v2 processor @ 2.3 GHz
  - Dual-socket – 8 cores per socket
  - 32 KB L1 cache, 256 KB L2 cache
  - DRAM – 59.6 GB/s Read bandwidth, 32.9 GB/s Write bandwidth

Comparison with Baselines: Execution Time



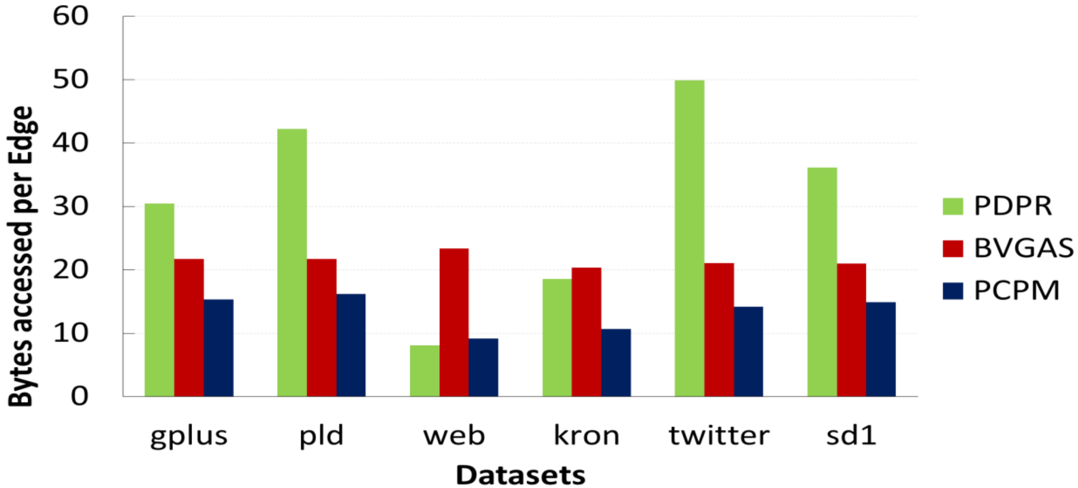
- Upto **4.1 ×** speedup over PDPR
- Upto **3.8 ×** speedup over BVGAS

Dataset	PDPR Total Time(s)	BVGAS Scatter Time(s)	BVGAS Gather Time(s)	BVGAS Total Time(s)	PCPM Scatter Time(s)	PCPM Gather Time(s)	PCPM Total Time(s)
gplus	0.44	0.26	0.12	0.38	0.06	0.1	0.16
pld	0.68	0.33	0.15	0.48	0.09	0.13	0.22
web	0.21	0.58	0.23	0.81	0.04	0.17	0.21
kron	0.65	0.5	0.22	0.72	0.07	0.18	0.25
twitter	1.83	0.79	0.32	1.11	0.18	0.27	0.45
sd1	1.97	1.07	0.42	1.49	0.24	0.35	0.59

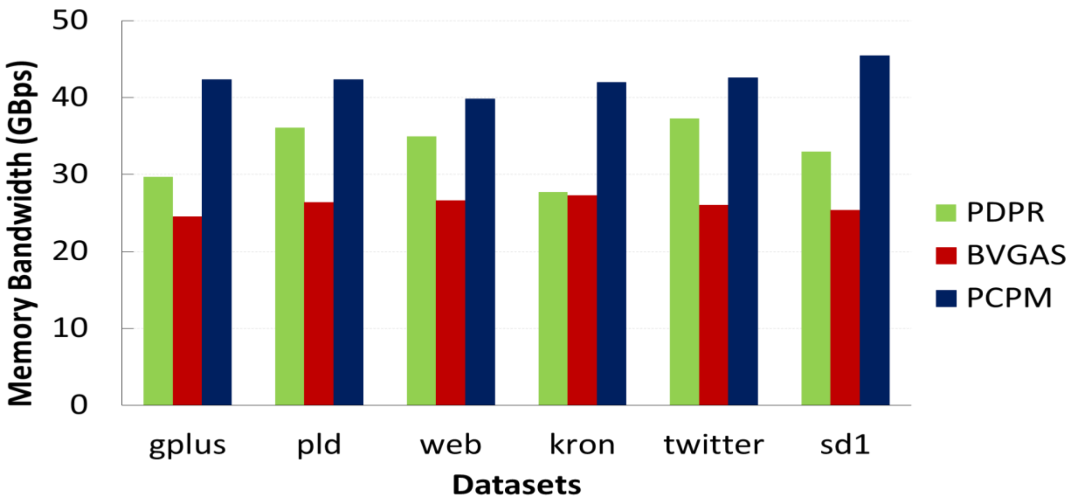
Table: Execution Time of 1 PageRank Iteration

- Average **5 ×** speedup in the *Scatter* phase
- Radically faster than BVGAS for high locality *web* graph

Comparison with Baselines: DRAM Performance



- Average $\mathbf{1.7} \times$ reduction in comm. volume over BVGAS
- Average $\mathbf{2.2} \times$ reduction in comm. volume over PDPR



- For $sd1$, PCPM sustained BW $\approx \mathbf{77\%}$ of peak BW
- Average $\mathbf{1.6} \times$ higher bandwidth than BVGAS

Comparison with Baselines: Effect of Locality

- $Orig$ → graph with original node labeling
- $GOrder$ → graph with GOrder[1] node labeling
  - Increased spatial locality among node neighbors

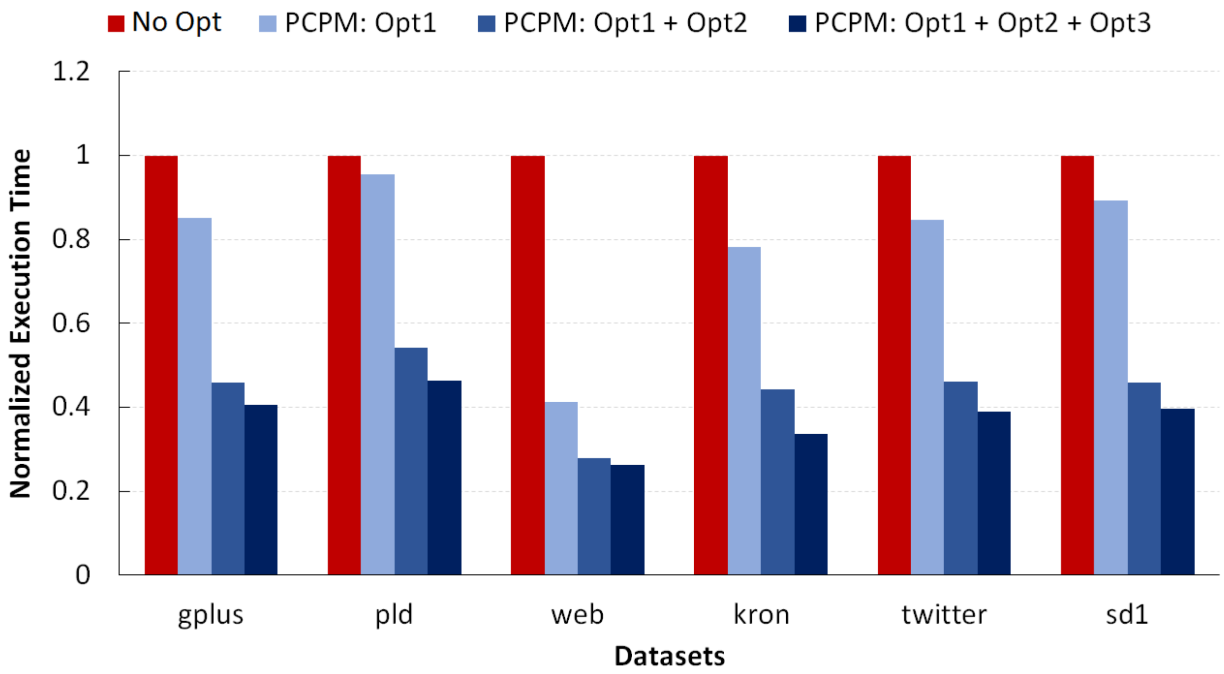
	PDPR		BVGAS		PCPM	
Dataset	*Orig*	*GOrder*	*Orig*	*GOrder*	*Orig*	*GOrder*
gplus	13.1	7.4	9.3	9.3	6.6	5.1
pld	24.5	10.7	12.6	12.5	9.4	6.1
web	7.5	7.6	21.6	21.3	8.5	8.4
kron	18.1	10.8	19.9	19.5	10.4	7.5
twitter	68.2	31.6	28.8	28.2	19.4	13.4
sd1	65.1	23.8	37.8	37.8	26.9	15.6

Table: PDPR and PCPM benefit from optimized node labeling

1. Wei, Hao, et al. "Speedup graph processing by graph ordering." *Proceedings of the 2016 International Conference on Management of Data*. ACM, 2016.

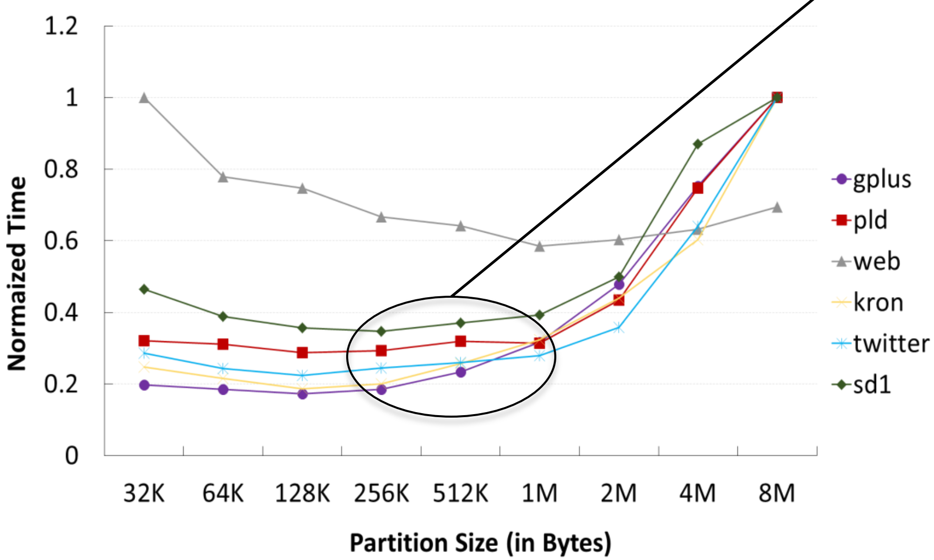
PCPM: Effect of Optimizations

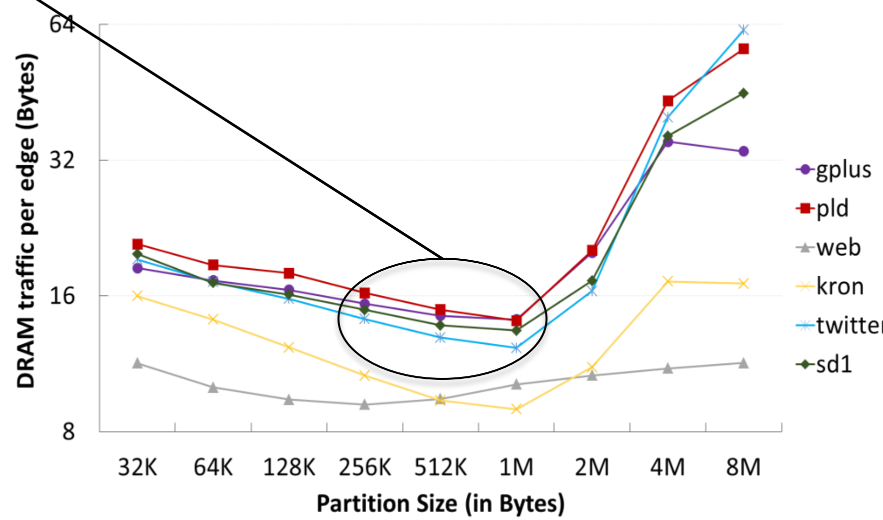
- Opt 1 → Partition-centric Update Propagation
- Opt 2 → PNG Data Layout
- Opt 3 → Branch Avoidance in $Gather$



PCPM: Effect of Partition Size

- $\uparrow$ partition size $\rightarrow \uparrow r$, $\downarrow$ DRAM traffic
- $\uparrow$ partition size beyond cache capacity $\rightarrow$ cache misses, sudden $\uparrow$ in DRAM traffic
- $256KB \leq$ size $\leq 1MB \rightarrow$ DRAM traffic $\downarrow$, execution time $\uparrow$
  - Vertex accesses served by slower L3 cache





Pre-processing Time

- Pre-processing → compute bin sizes, PNG construction
- Optimizations
  - Pre-process all partitions in parallel
  - Exploit overlap in bin size computation and PNG construction
- Result → very small overhead
  - Easily amortized over few PageRank iterations

Dataset	PCPM	BVGAS	PDPR
gplus	0.25s	0.1s	0s
pld	0.32s	0.15s	0s
web	0.26s	0.18s	0s
kron	0.43s	0.22s	0s
twitter	0.7s	0.27s	0s
sd1	0.95s	0.32s	0s

Table: Pre-processing time of different methodologies

Outline

- Introduction
- Partition-centric Processing Methodology
- Analytical Evaluation
- Experimental Results
- **Generalization**
- Conclusion

Generalization

- PageRank is an example
- PCPM for processing weighted graphs
  - possible programming model for graph analytics

Updates	Dest. ID	Edge Wt.
PR[6]	2	w_{62}
PR[7]	0	w_{70}
	1	w_{71}
	2	w_{72}

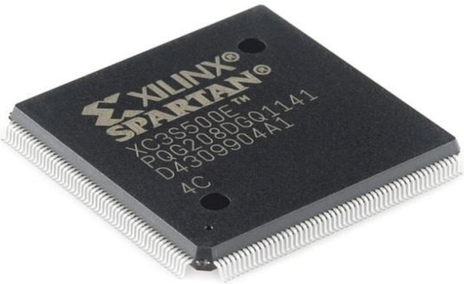
Bin 0

- Extendible to generic SpMV (non-square matrices) computation
  - partition rows and columns separately
  - parallelize $Scatter$ over column partitions
  - parallelize $Gather$ over row partitions

Generalization

- PCPM optimizations are generic software techniques
  - not specific to the multicore platform used

- Can be ported to FPGAs and GPUs as well
  - FPGAs → store vertex data in BRAM
  - GPUs → store vertex data in shared memory
  - user-controlled on-chip memories even more suitable



Outline

- Introduction
- Partition-centric Processing Methodology
- Analytical Evaluation
- Experimental Results
- Generalization
- **Conclusion**

Conclusion

- Proposed novel Partition-centric method for PageRank
- Developed optimizations to
  - Reduce volume of DRAM traffic
  - Enhance sustained DRAM bandwidth
- Comparison with state-of-the-art on multicore
  - Average $\mathbf{2.7}\times$ increase in throughput
  - Average $\mathbf{1.7}\times$ reduction in DRAM communication
  - Average $\mathbf{1.6}\times$ higher sustained memory bandwidth
- Can be extended to
  - Weighted graphs and generic SpMV
  - Other platforms such as GPUs and FPGAs etc.

USC University of Southern California

Code can be found at:
https://github.com/kartiklakhotia/pcpm

Comments & Questions

Thank you

projectsharp.usc.edu [DARPA HIVE]