

Multicore Triangle Computations Without Tuning

Julian Shun and Kanat Tangwongsan

Presentation is based on paper published in International
Conference on Data Engineering (ICDE), 2015

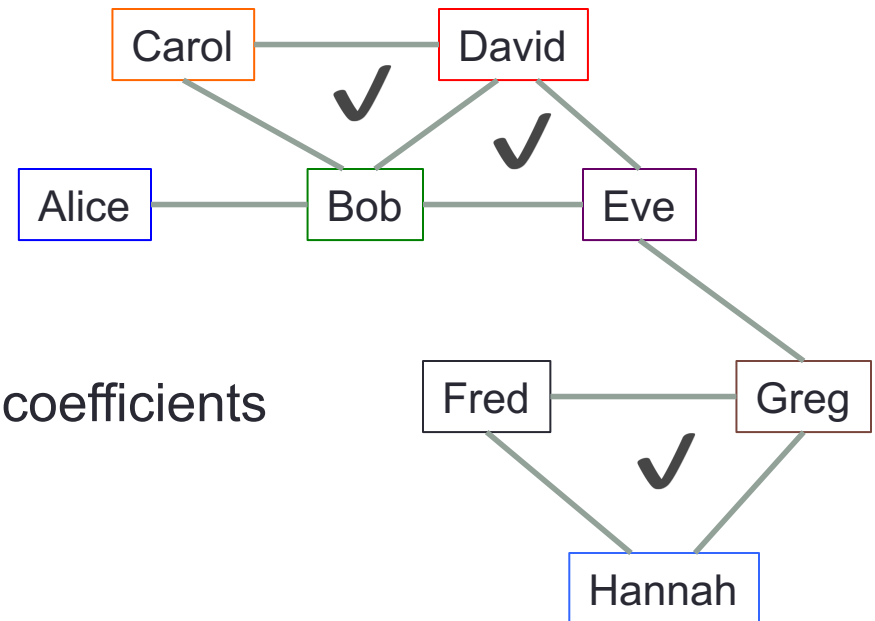
Triangle Computations

- Triangle Counting

Count = 3

- Other variants:

- Triangle listing
- Local triangle counting/clustering coefficients
- Triangle enumeration
- Approximate counting
- Analogs on directed graphs



- Numerous applications...

- Social network analysis, Web structure, spam detection, outlier detection, dense subgraph mining, 3-way database joins, etc.

Need fast triangle computation algorithms!

Sequential Triangle Computation Algorithms

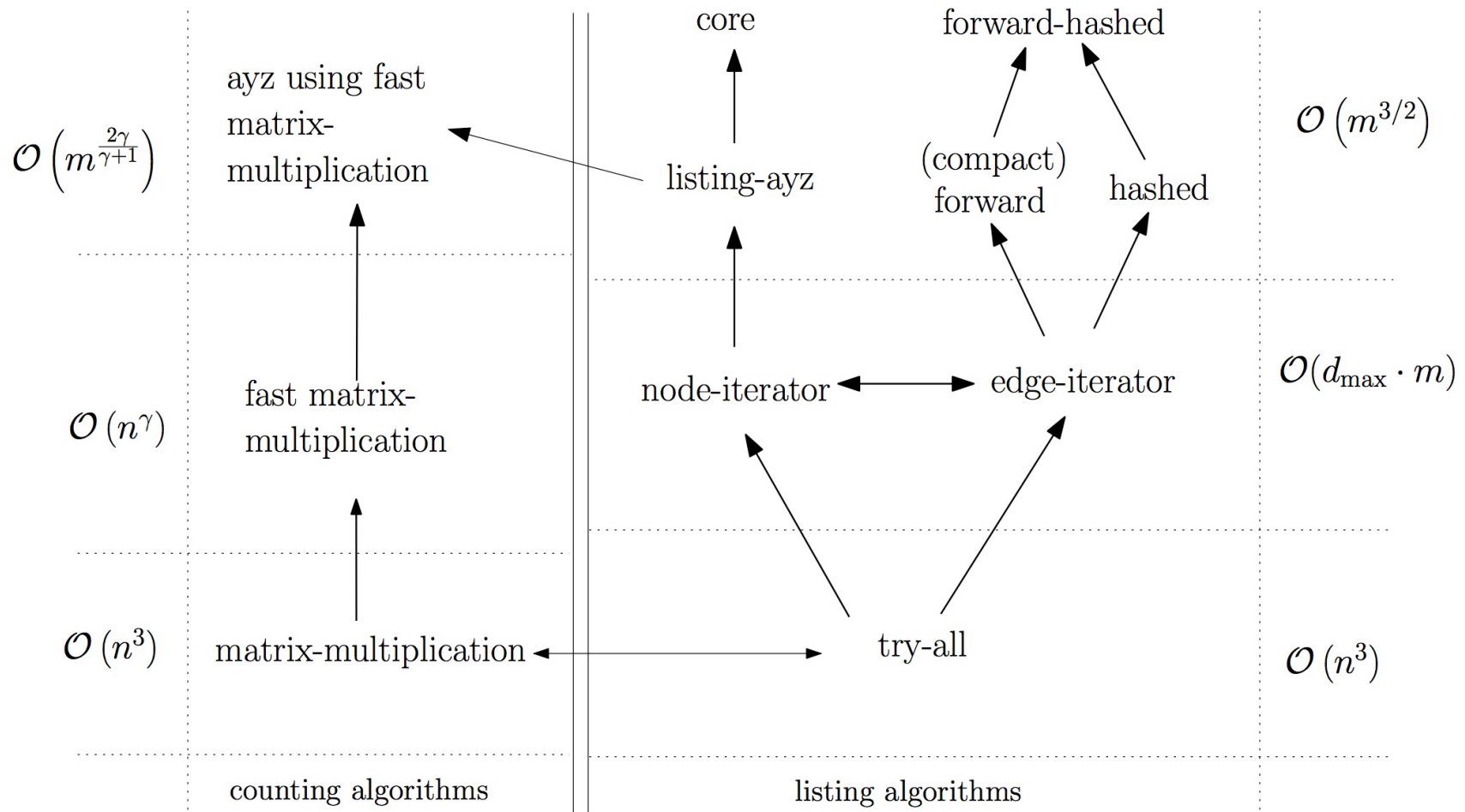
$V = \#$ vertices

$E = \#$ edges

- Sequential algorithms for exact counting/listing
 - Naïve algorithm of trying all triplets
 $O(V^3)$ work
 - Node-iterator algorithm [Schank]
 $O(VE)$ work
 - Edge-iterator algorithm [Itai-Rodeh]
 $O(VE)$ work
 - Tree-lister [Itai-Rodeh], forward/compact-forward [Schank-Wagner, Lapaty]
 $O(E^{1.5})$ work
- Sequential algorithms via matrix multiplication
 - $O(V^{2.37})$ work compute A^3 , where A is the adjacency matrix
 - $O(E^{1.41})$ work [Alon-Yuster-Zwick]
 - These require superlinear space

Sequential Triangle Computation Algorithms

Source: “Algorithmic Aspects of Triangle-Based Network Analysis”, Dissertation by Thomas Schank



What about parallel algorithms?

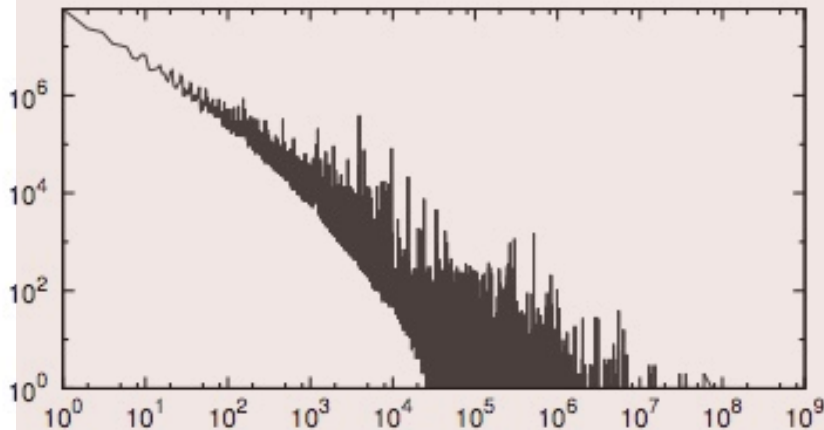
Parallel Triangle Computation Algorithms

- Most designed for distributed memory
 - MapReduce algorithms [Cohen '09, Suri-Vassilvitskii '11, Park-Chung '13, Park et al. '14]
 - MPI algorithms [Arifuzzaman et al. '13, Graphlab]
- *What about shared-memory multicore?*
 - **Multicores are everywhere!**
 - Node-iterator algorithm [Green et al. '14]
 - $O(VE)$ work in worst case
- *Can we obtain an $O(E^{1.5})$ work shared-memory multicore algorithm?*

Triangle Computation: Challenges for Shared Memory Machines

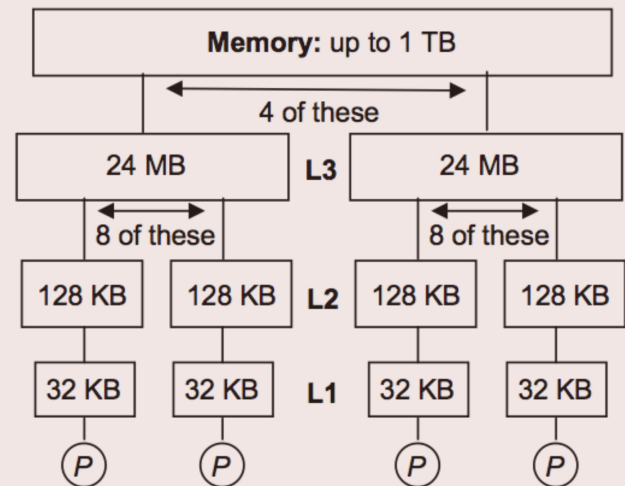
1

*Irregular
computation*



2

*Deep memory
hierarchy*



External-Memory and Cache-Oblivious Triangle Computation

- All previous algorithms are sequential
- External-memory (cache-aware) algorithms
 - Natural-join $O(E^3/(M^2 B))$ I/O's
 - Node-iterator [Dementiev '06] $O((E^{1.5}/B) \log_{M/B}(E/B))$ I/O's
 - Compact-forward [Menegola '10] $O(E + E^{1.5}/B)$ I/O's
 - [Chu-Cheng '11, Hu et al. '13] $O(E^2/(MB) + \#\text{triangles}/B)$ I/O's
- External-memory and cache-oblivious
 - [Pagh-Silvestri '14] $O(E^{1.5}/(M^{0.5} B))$ I/O's or cache misses
- *Parallel cache-oblivious algorithms?*

Our Contributions

1 *Parallel Cache-Oblivious Triangle Counting Algs*

Algorithm	Work	Depth	Cache Complexity
TC-Merge	$O(E^{1.5})$	$O(\log^2 E)$	$O(E + E^{1.5}/B)$
TC-Hash	$O(V \log V + \alpha E)$	$O(\log^2 E)$	$O(\text{sort}(V) + \alpha E)$
Par. Pagh-Silvestri	$O(E^{1.5})$	$O(\log^3 E)$	$O(E^{1.5}/(M^{0.5} B))$

V = # vertices
 M = cache size

E = # edges
 B = line size

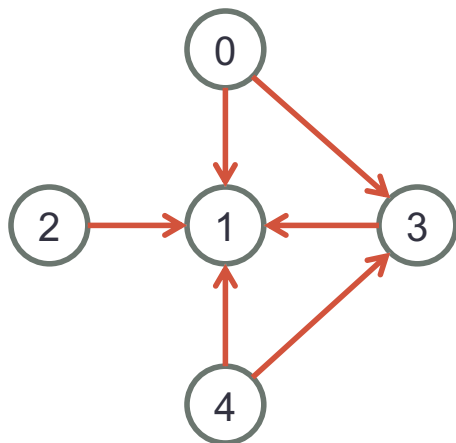
α = arboricity (at most $E^{0.5}$)
 $\text{sort}(n) = (n/B) \log_{M/B}(n/B)$

2 *Extensions to Other Triangle Computations: Enumeration, Listing, Local Counting/Clustering Coefficients, Approx. Counting, Variants on Directed Graphs*

3 *Extensive Experimental Study*

Sequential Triangle Counting (Exact)

(Forward/compact-forward algorithm)



Rank vertices by degree (sorting)
Return $A[v]$ for all v storing higher ranked neighbors

1

for each vertex v :

for each w in $A[v]$:

count += intersect($A[v]$, $A[w]$)

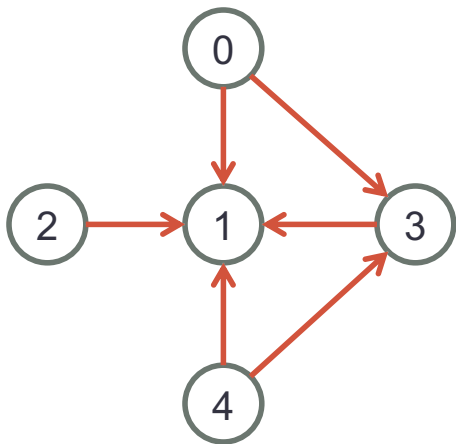
2

Gives all triangles (v, w, x) where
 $\text{rank}(v) < \text{rank}(w) < \text{rank}(x)$

Work = $O(E^{1.5})$

[Schank-Wagner '05, Latapy '08]

Proof of $O(E^{1.5})$ work bound when intersect uses merging



Rank vertices by degree (sorting)
Return $A[v]$ for all v storing higher ranked neighbors

1

for each vertex v :

for each w in $A[v]$:

count += intersect($A[v]$, $A[w]$)

2

- Step 1: $O(E+V \log V)$ work
- Step 2:
 - For each edge (v,w) , intersect does $O(d^+(v) + d^+(w))$ work
 - For all v , $d^+(v) \leq 2E^{0.5}$
 - If $d^+(v) > 2E^{0.5}$, each of its higher ranked neighbors also have degree $> 2E^{0.5}$ and total number of directed edges $> 4E$, a contradiction
 - Total work = $E * O(E^{0.5}) = O(E^{1.5})$

Parallel Triangle Counting (Exact)

Step 1

Work = $O(E+V \log V)$

Depth = $O(\log^2 V)$

Cache = $O(E+\text{sort}(V))$

Parallel sort
and filter



Rank vertices by degree (sorting)

Return $A[v]$ for all v storing higher ranked neighbors

1

parallel_for each vertex v :

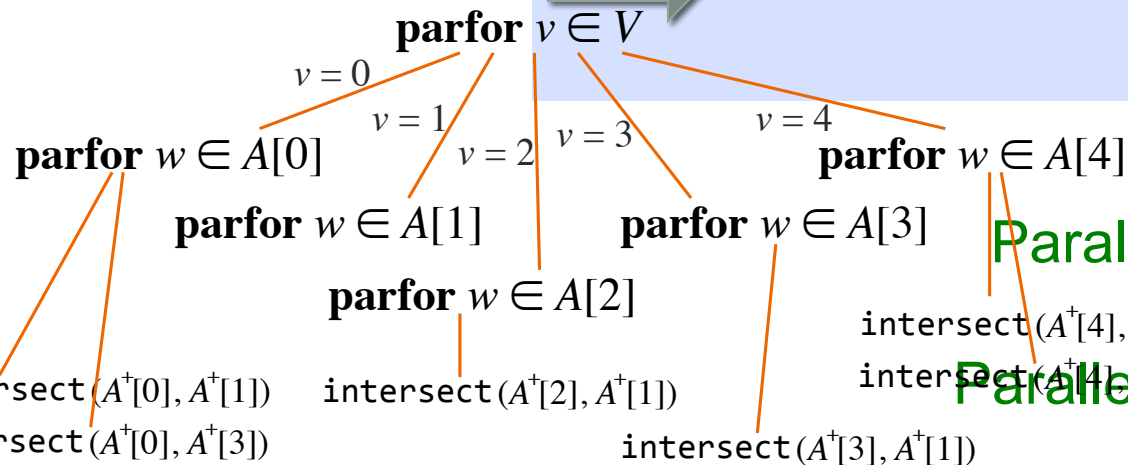
parallel_for each w in $A[v]$:

Parallel reduction



count += intersect($A[v]$, $A[w]$)

2



Parallel merge (TC-Merge)

Parallel hash table (TC-Hash)

Safe to
run all in
parallel



TC-Merge and TC-Hash Details

parallel_for each vertex v :

parallel_for each w in $A[v]$:

Parallel reduction



count += intersect($A[v]$, $A[w]$)

2

Step 2: TC-Merge

Work = $O(E^{1.5})$

Depth = $O(\log^2 E)$

Cache = $O(E + E^{1.5}/B)$

Step 2: TC-Hash

Work = $O(\alpha E)$

Depth = $O(\log E)$

Cache = $O(\alpha E)$

(α = arboricity (at most $E^{0.5}$))

Parallel merge (TC-Merge)
or

Parallel hash table (TC-Hash)

- TC-Merge

- Preprocessing: sort adjacency lists
- Intersect: use a parallel and cache-oblivious merge based on divide-and-conquer [Blelloch et al. '10, Blelloch et al. '11]

- TC-Hash

- Preprocessing: for each vertex, create parallel hash table storing edges [Shun-Blelloch '14]
- Intersect: scan smaller list, querying hash table of larger list in parallel

Comparison of Complexity Bounds

Algorithm	Work	Depth	Cache Complexity
TC-Merge	$O(E^{1.5})$	$O(\log^2 E)$	$O(E + E^{1.5}/B)$ (<i>oblivious</i>)
TC-Hash	$O(V \log V + \alpha E)$	$O(\log^2 E)$	$O(\text{sort}(V) + \alpha E)$ (<i>oblivious</i>)
Par. Pagh-Silvestri	$O(E^{1.5})$	$O(\log^3 E)$	$O(E^{1.5}/(M^{0.5} B))$ (<i>oblivious</i>)
Chu-Cheng '11, Hu et al. '13	$O(E \log E + E^2/M + \alpha E)$		$O(E^2/(MB) + \#\text{triangles}/B)$ (<i>aware</i>)
Pagh-Silvestri '14	$O(E^{1.5})$		$O(E^{1.5}/(M^{0.5} B))$ (<i>oblivious</i>)
Green et al. '14	$O(VE)$	$O(\log E)$	

V = # vertices
 M = cache size

E = # edges
 B = line size

α = arboricity (at most $E^{0.5}$)
 $\text{sort}(n) = (n/B) \log_{M/B}(n/B)$

Our Contributions

1 *Parallel Cache-Oblivious Triangle Counting Algs*

Algorithm	Work	Depth	Cache Complexity
TC-Merge	$O(E^{1.5})$	$O(\log^2 E)$	$O(E + E^{1.5}/B)$
TC-Hash	$O(V \log V + \alpha E)$	$O(\log^2 E)$	$O(\text{sort}(V) + \alpha E)$
Par. Pagh-Silvestri	$O(E^{1.5})$	$O(\log^3 E)$	$O(E^{1.5}/(M^{0.5} B))$

V = # vertices
 M = cache size

E = # edges
 B = line size

α = arboricity (at most $E^{0.5}$)
 $\text{sort}(n) = (n/B) \log_{M/B}(n/B)$

2 *Extensions to Other Triangle Computations:* *Enumeration, Listing, Local Counting/Clustering Coefficients* *Approx. Counting, Variants on Directed Graphs*

3 *Extensive Experimental Study*

Extensions of Exact Counting Algorithms

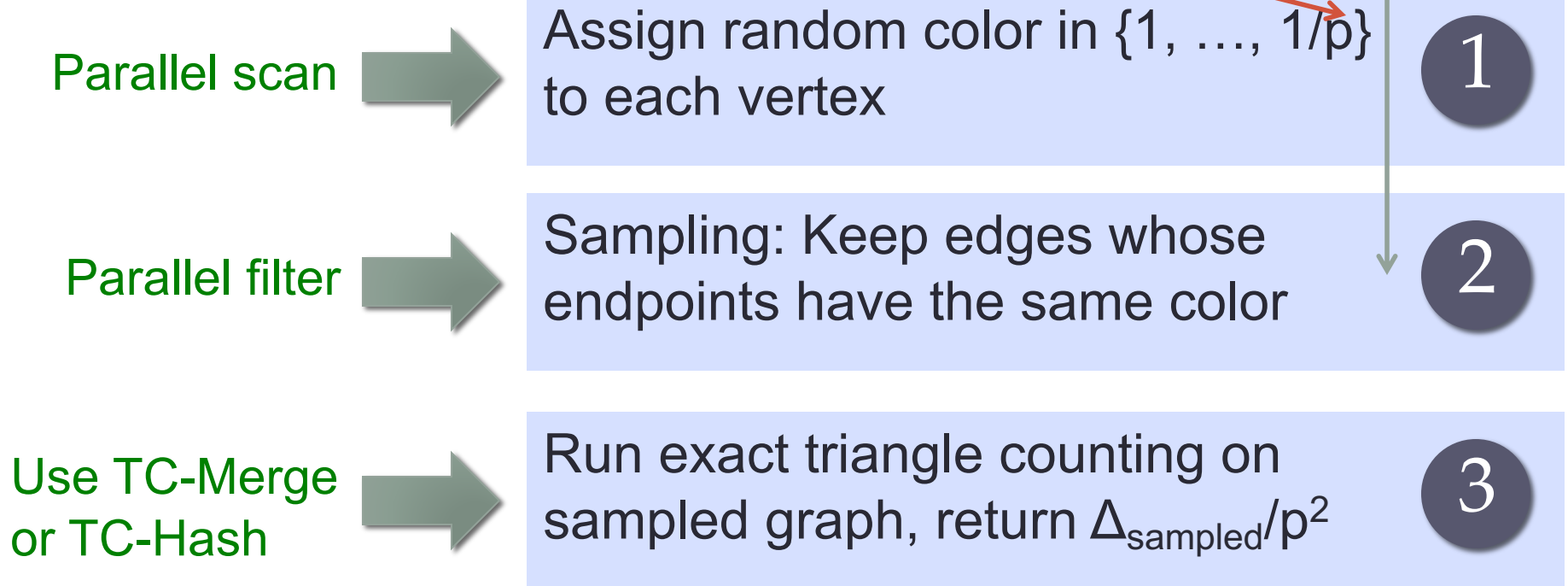
- Triangle enumeration
 - Call **emit** function whenever triangle is found
 - **Listing**: add to hash table to list; return contents at the end
 - **Local counting/clustering coefficients**: atomically increment count of three triangle endpoints
- Directed triangle counting/enumeration
 - Keep separate counts for different types of triangles
- Approximate counting
 - Use colorful triangle sampling scheme to create smaller sub-graph [Pagh-Tsourakakis '12]
 - Run TC-Merge or TC-Hash on sub-graph with pE edges ($0 < p < 1$) and return $\#triangles/p^2$ as estimate

Approximate Counting

Expected # edges = pE

- Colorful triangle counting [Pagh-Tsourakakis '12]

Sampling rate: $0 < p < 1$



Steps 1 & 2

Work = $O(E)$

Depth = $O(\log E)$

Cache = $O(E/B)$

Step 3: TC-Merge

Work = $O((pE)^{1.5})$

Depth = $O(\log^2 E)$

Cache = $O(pE + (pE)^{1.5}/B)$

Step 3: TC-Hash

Work = $O(V \log V + \alpha pE)$

Depth = $O(\log E)$

Cache = $O(\text{sort}(V) + \alpha pE)$

Our Contributions

1 *Parallel Cache-Oblivious Triangle Counting Algs*

Algorithm	Work	Depth	Cache Complexity
TC-Merge	$O(E^{1.5})$	$O(\log^2 E)$	$O(E + E^{1.5}/B)$
TC-Hash	$O(V \log V + \alpha E)$	$O(\log^2 E)$	$O(\text{sort}(V) + \alpha E)$
Par. Pagh-Silvestri	$O(E^{1.5})$	$O(\log^3 E)$	$O(E^{1.5}/(M^{0.5} B))$

V = # vertices
 M = cache size

E = # edges
 B = line size

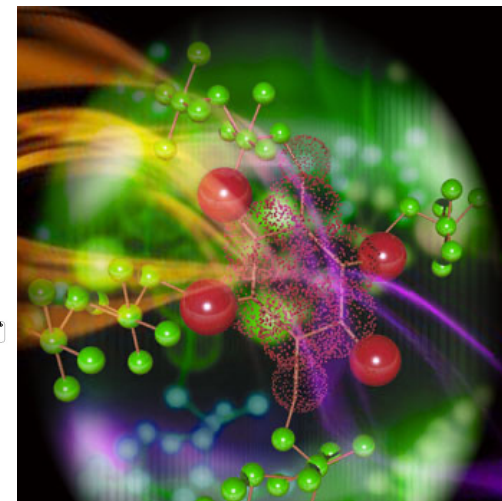
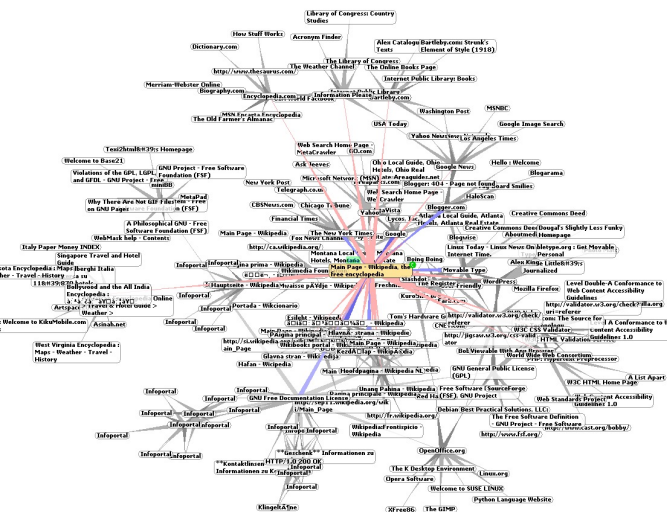
α = arboricity (at most $E^{0.5}$)
 $\text{sort}(n) = (n/B) \log_{M/B}(n/B)$

2 ~~*Extensions to Other Triangle Computations:*~~
~~*Enumeration, Listing, Local Counting/Clustering Coefficients,*~~
~~*Approx. Counting, Variants on Directed Graphs*~~

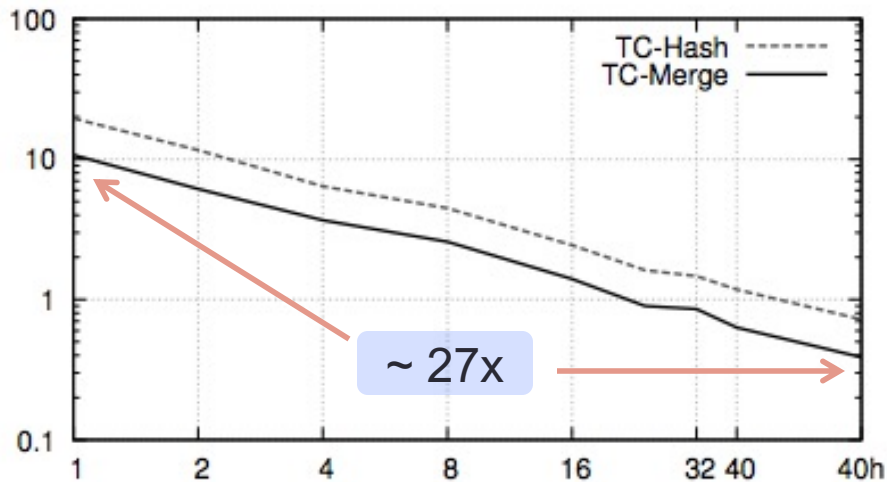
3 *Extensive Experimental Study*

Experimental Setup

- Implementations using Intel Cilk Plus
- 40-core Intel Nehalem machine (with 2-way hyper-threading)
 - 4 sockets, each with 30MB shared L3 cache, 256KB private L2 caches
- Sequential TC-Merge as baseline (faster than existing sequential implementations)
- Other multicore implementations: Green et al. and GraphLab
- Our parallel Pagh-Silvestri algorithm was not competitive
- Variety of real-world and artificial graphs

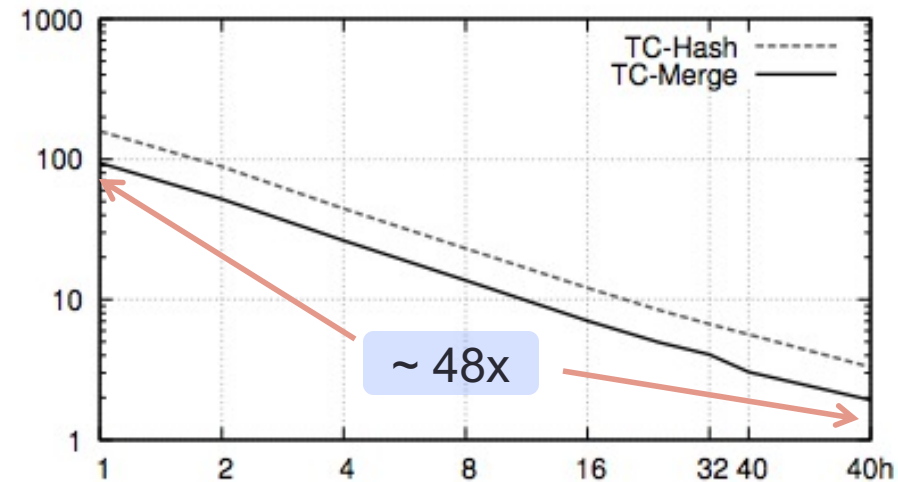


Both TC-Merge and TC-Hash scale well with # of cores:



LiveJournal

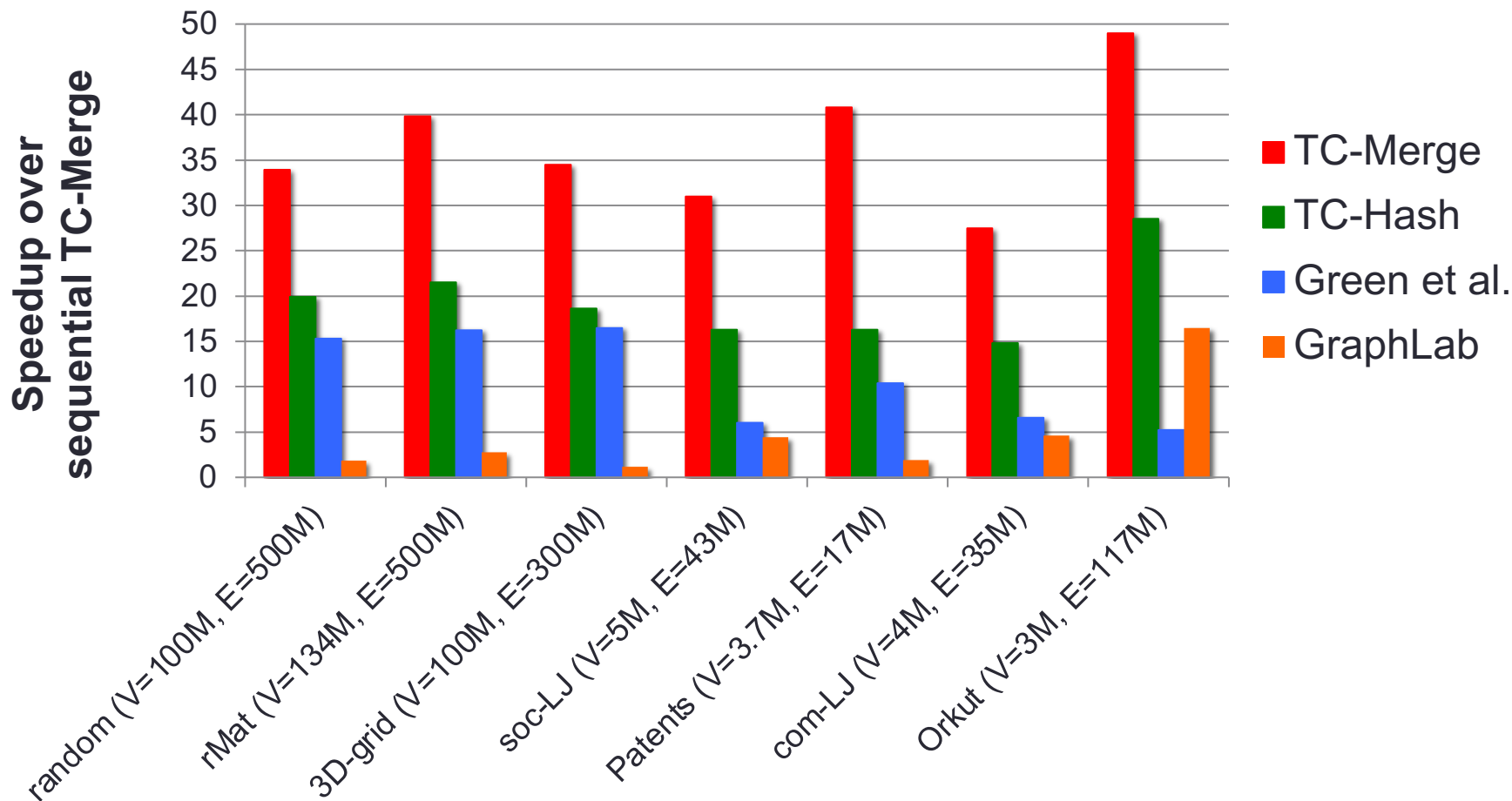
4M vtxes, 34.6M edges



Orkut

3M vtxes, 117M edges

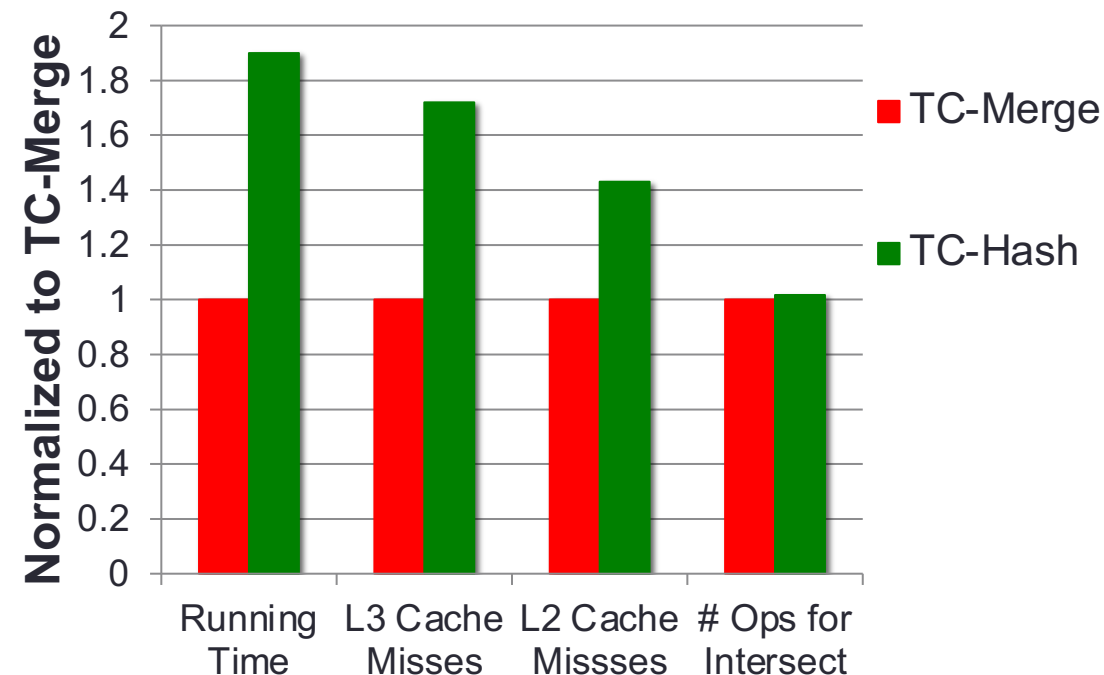
40-core (with hyper-threading) Performance



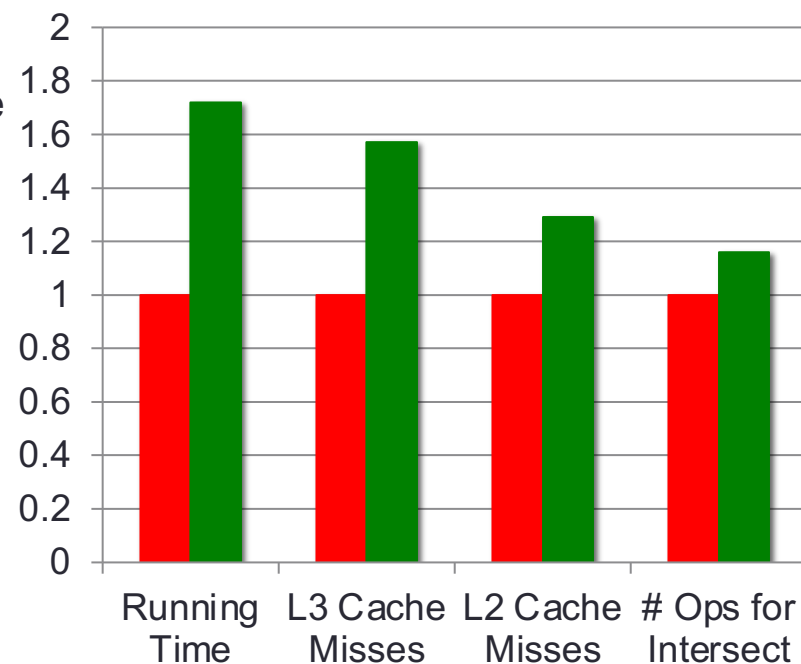
- TC-Merge always faster than TC-Hash (by 1.3—2.5x)
- TC-Merge always faster than Green et al. or GraphLab (by 2.1—5.2x)

Why is TC-Merge faster than TC-Hash?

soc-LJ



Orkut



- TC-Hash less cache-efficient than TC-Merge
- Running time more correlated with cache misses than work

Comparison to existing counting algs.

Twitter graph (41M vertices, 1.2B undirected edges, 34.8B triangles)

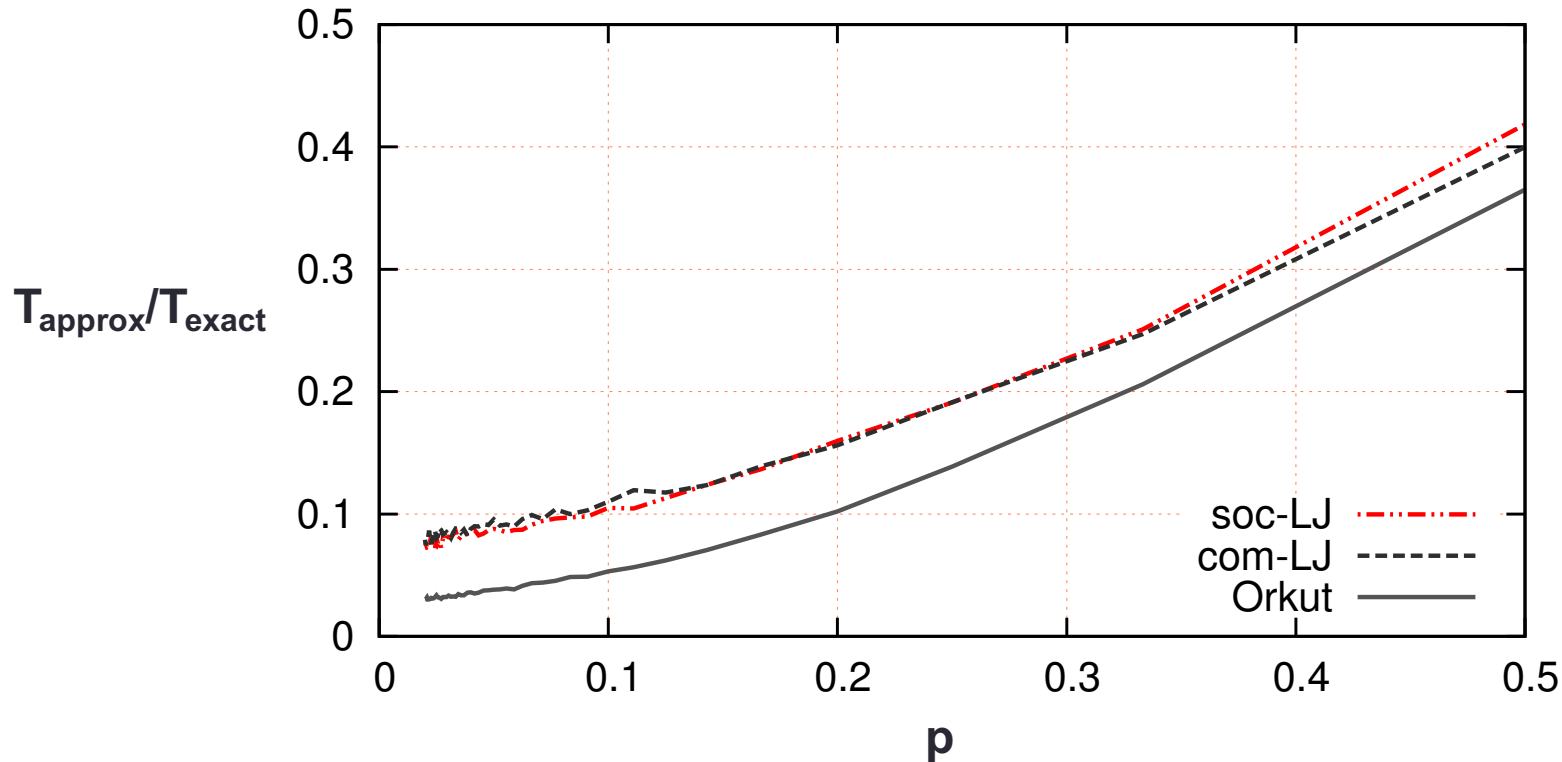
- **Yahoo graph** (1.4B vertices, 6.4B edges, 85.8B triangles)
on 40 cores: **TC-Merge takes 78 seconds**
 - Approximate counting algorithm achieves **99.6% accuracy in 9.1 seconds**

Shared vs. distributed memory costs

- Amazon EC2 pricing
 - Captures purchasing costs, maintenance/operating costs, energy costs

Triangle Counting (Twitter)	Our algorithm	GraphLab	GraphLab
Running Time	0.932 min	3 min	1.5 min
Machine	40-core (256 GB memory)	40-core (256 GB memory)	64 x 16-core
Approx. EC2 pricing	< \$4/hour	< \$4/hour	64 x \$0.928/hour
Overall cost	< \$0.062	< \$0.2	\$1.49

Approximate counting



$p=1/25$	Accuracy	T_{approx}	$T_{\text{approx}}/T_{\text{exact}}$
Orkut (V=3M, E=117M)	99.8%	0.067sec	0.035
Twitter (V=41M, E=1.2B)	99.9%	2.4sec	0.043
Yahoo (V=1.4B, E=6.4B)	99.6%	9.1sec	0.117

Conclusion

Algorithm	Work	Depth	Cache Complexity
TC-Merge	$O(E^{1.5})$	$O(\log^2 E)$	$O(E + E^{1.5}/B)$
TC-Hash	$O(V \log V + \alpha E)$	$O(\log^2 E)$	$O(\text{sort}(V) + \alpha E)$
Par. Pagh-Silvestri	$O(E^{1.5})$	$O(\log^3 E)$	$O(E^{1.5}/(M^{0.5} B))$

- Simple multicore algorithms for triangle computations are provably work-efficient, low-depth, and cache-efficient
- Implementations require no load-balancing or tuning for cache
- Experimentally outperforms existing multicore and distributed algorithms
- Future work: Design a practical parallel algorithm achieving $O(E^{1.5}/(M^{0.5} B))$ cache complexity



GraphChallenge

