H-INDEX: Hash-Indexing for Parallel Triangle Counting on GPUs

Santosh Pandey Xiaoye Sherry Li+ Aydin Buluc+ Jiejun Xu* Hang Liu
Stevens Institute of Technology +: Lawrence Berkeley National Laboratory *: HRL Laboratories

***Abstract*—Triangle counting is a graph algorithm that calculates the number of triangles involving each vertex in a graph. Briefly, a triangle encompasses three vertices from a graph, where every vertex possesses at least one incidental edge to the other two vertices from the triangle. Consequently, list intersection, which identifies the incidental edges, becomes the core algorithm for triangle counting. At the meantime, attracted by the enormous parallel computing potential of Graphics Processing Units (GPUs), numerous efforts have been devoted to deploy triangle counting algorithms on GPUs.**

While state-of-the-art intersection algorithms, such as merge-path and binary-search, perform well on traditional multi-core CPU systems, deploying them on massively parallel GPUs turns out to be challenging. In particular, merge-path based approach experiences the hardship of evenly distributing the workload across vast GPU threads and irregular memory accesses. Binary-search based approach often suffers from the potential problem of high time complexity. Furthermore, both approaches require sorted neighbor lists from the input graphs, which involves nontrivial preprocessing overhead. To this end, we introduce H-INDEX, a hash-indexing assisted triangle counting algorithm that overcomes all the aforementioned shortcomings. Notably, H-INDEX achieves 141.399 billion TEPS computing rate on a Protein K-mer V2a graph with 64 GPUs. To the best of our knowledge, this is the first work that advances triangle counting beyond the 100 billion TEPS rate.

I. INTRODUCTION

Triangle counting gains popularity from social network analysis, where it is exploited to detect communities and measure the corresponding cohesiveness [12], [23]. Particularly, one can use triangle counting to measure the robustness of the graph - the deletion of an edge in a triangle will not result in the disconnectedness of these three vertices. And triangle counting is also a primary routine for local and global clustering coefficient analysis of a network [20], [21], [13].

Triangle counting can be formulated into problems from two different domains, that is, graph computing and graph mining. The former domain further encompasses vertex and edge-centric approaches [14], [19], [16], [7]. The vertex-centric one iterates through each vertex, fetches its neighbors, and counts the pair of neighbors that possess intermediate edge(s), while **the edge-centric option iterates through each edge and counts the common neighbors originated from the source and destination vertices of the edge**. In graph mining domain, one will first find all open triangles through tree-based prune and subsequently verify whether there exists an edge from the graph that can complete this triangle [5].

Given edge-centric approach outperforms the other alternatives [12], [11], [22], intersecting two sorted neighbor lists to count the common neighbors becomes the key for triangle counting. Particularly, there are two mainstream designs on this track, that is, merge-path and binary-search based methods. Assuming two neighbor lists - M and N - are at size of $|M|$ and $|N|$, where $|M| \leq |N|$, the time complexities of merge-path and binary-search based designs are $\mathcal{O}(|M| + |N|)$ [8], [9] and $\mathcal{O}(|M| \cdot log|N|)$ [12], respectively. Therefore, binary-search based approach is deemed more efficient when $|M| >> |N|$. However, the popular edge orientation method [10] will substantially shrink the difference between M and N, leading merge-path to be more efficient, in terms of time complexity, than binary-search based alternative. Surprisingly, TriCore [12] demonstrates that binary-search based design is significantly faster than the merge-path based one, blaming the overhead of properly distributing the workload across GPU threads in merge-path based method, as well as its unfriendly memory access patterns.

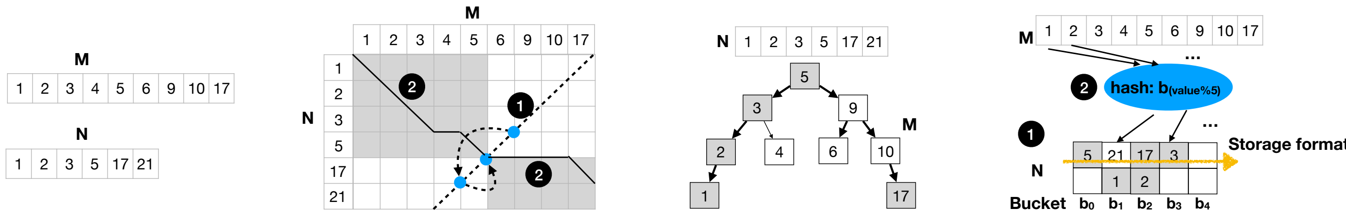
For the completeness of related work review, we also briefly discuss other approaches. For instance, matrix multiplication based design [24], [4], linear algebra-based [17] and subgraph matching [23] based approaches are also explored, which however often require more preprocessing efforts or memory space. Further, [6] implements the bitmap-based intersection but requires atomic operations to update the bitmap in parallel and also suffers from strided memory access pattern.

This work proposes H-INDEX, a hash-indexing based approach to accelerate parallel triangle counting. Particularly, H-INDEX comes with the following two contributions.

First, inspired by the observation that only items that go to the same bucket will have the chance of being identical, we use hashing to narrow down the search space for intersection. Briefly, H-INDEX uses the shorter neighbor list N to construct a collection of buckets for the longer neighbor list M to search against. Particularly, H-INDEX first uses hash to distribute various neighbors of N into a collection of buckets. Afterwards, H-INDEX hashes each neighbor of the longer neighbor list - search key - to a corresponding bucket. Eventually, we will use this search key to search through that bucket to determine whether there exists an identical neighbor from N in this bucket. Once an identical neighbor is found, a triangle is identified. Note, we use N to construct the collection of buckets, hoping shorter neighbor list will potentially experience lower collisions.

Second, to coalesce the memory access, we interleave hash entries from all buckets of the shorter neighbor list. That is, the first elements from all buckets are stored consecutively,

978-1-7281-5020-8/19/$31.00 ©2019 IEEE



(a) Input list (b) Merge-path method (c) Binary-search method (d) H-INDEX standalone approach

Fig. 1: Various triangle counting algorithms for the sorted (a) input lists. Particularly, we assume two threads working on the intersection with (b) merge-path based approach, (c) binary-search approach, (d) H-INDEX approach with memory friendly bucket storage format.

subsequently the second elements and etc. We propose this design because of the fact that during searching for common neighbors, all threads (of a Warp or CTA in GPU) will start from the first element in various buckets. In this context, storing all the elements of the same bucket contiguously will introduce strided memory access.

It is important to note that H-INDEX is not the first attempt that uses hashing mechanism for triangle counting - [22], [25] also explore this direction. However, H-INDEX stands out in two ways: first, H-INDEX proposes to hash the shorter neighbor list and use the longer one as the search key to reduce the potential of collisions. Second, H-INDEX introduces a memory friendly bucket placement strategy for GPUs. For comparing the performance, we compute the rate of Traversed Edges Per Second (TEPS) for each graph.

Highlights. H-INDEX retains a maximum of 141.399 billion TEPS on the Protein K-mer V2a graph with 64 GPUs. **To the best of our knowledge, H-INDEX is the first work to achieve beyond 100 billion TEPS computing rate for the triangle counting problem.** It is also important to note that H-INDEX avoids sorting the neighbor lists during preprocessing, which is essential and time consuming for both binary-search and merge-path based approaches.

The rest of this paper is organized as follows. Section II presents the conventional triangle counting algorithms. Section III describe the novel H-INDEX designs. Section IV studies the performance of H-INDEX across all datasets presented in graph challenge and Section V concludes.

II. Triangle Counting Algorithms

This section discusses the designs of basic triangle counting algorithms on GPUs.

Merge-path based triangle counting first performs a binary search on the dotted line in Figure 1(b) (❶) to partition the intersection task into left and right shadow sides with roughly similar workloads. The eventual partition will be 1 - 5 of M and N for left side and the rest for right side. Note, this design also ensures the entries of left side of M will not interfere with the right side of N, and vice verse. Afterwards, two threads will work on the left and right shadows in parallel (❷). In Figure 1(b), $\setminus$ means a common neighbor is found between M and N.

Binary-search based triangle counting treats the longer input list M as the binary tree and the shorter list - N - as the search key list. During intersection, as shown in Figure 1(c), each search key will descend the binary search tree until either a matching value is found or the leaf vertex of the tree is reached. The rule of descending is that we will take a left branch if the search key is smaller than the current value in the binary search and right branch otherwise. For instance, for search key "1", we will first check whether it equals the root "5". Since "1" is smaller than "5", we will take the left branch. This process continues until we find a match.

Motivation. H-INDEX is motivated by the fact that both merge-path and binary-search based methods go through noticeable drawbacks as follows:

Merge-path based approach excels at maintaining the low time complexity of $\mathcal{O}(M+N)$ but suffers from two shortcomings. First, it requires nontrivial effort to correctly partition M and N into the top left and bottom right sublists as shown in Figure 1(b). Second, during intersection (❷), consecutive threads (i.e., threads 0 and 1) will have to work on far-apart data. That is, while thread 0 checks whether M[0] equals N[0], thread 1 will work on M[5] and N[5]. This leads to low memory throughput.

Binary-search based approach primarily suffers from two problems. First, the time complexity is relatively high if M and N are similarly large. For instance, if $M = N = 128$, the total number operations in binary search is 896 (i.e., $128 \times log128$). In contrast, merge-path only introduces 256 (i.e., $128 + 128$) operations. Second, the access of binary-search tree will become strided when descending the tree. For instance, one thread is working on search key 5 while the other on search key 17.

III. H-Index Design

This section discusses the proposed H-INDEX algorithm.

Figure 1(d) explains the H-INDEX standalone approach. Particularly, we will hash the shorter neighbor list N to construct five buckets, i.e., b_0 - b_4. Afterwards, we will iterate through the larger array M and hash each entry to the corresponding bucket. Eventually, a linear search is performed to see whether a similar neighbor from N exists in that bucket. Algorithm 1 shows the procedure for hash-based triangle counting.

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

Algorithm 1 Hash-based triangle counting

procedure TRIANGLECOUNT(G)
  for each edge (u,v) **do**
    if degree(u)$\leq$ degree(v) **then**
      shorterList $\leftarrow$ neighborList(u);
      longerList $\leftarrow$ neighborList(v);
    else
      shorterList $\leftarrow$ neighborList(v);
      longerList $\leftarrow$ neighborList(u);
    end if
    ht = HASH (shorterList);
    for each neighbor $\in$ longerList **do**
      bucket$\leftarrow$ HASH (neighbor);
      count = linearSearch (ht[bucket], neighbor);
    end for
    totalTriangle += count;
  end for
  return totalTriangle
end procedure

Low collision is essential for the success of H-INDEX. This is also the reason that we often choose the shorter array, i.e., N to hash and construct the buckets. Once a collision, unfortunately, happens, that is, one bucket contains more than one entry, e.g., b_1 and b_2 in Figure 1(d), we will need to conduct a linear search to check if a match is found when hashing M. For instance, when we are working on entry $M[0] = 1$, we need to do a linear search to find the common neighbor '1' from b_1. Given we can afford hundreds of hashing buckets in fast GPU shared memory, we can actually observe very few collisions. For vertices with long neighbor lists, collision is inevitable even with larger bucket size. However, a larger bucket size can help mitigate the impact.

For better memory access pattern, as shown in Figure 1(d), we will store various buckets from neighbor list N in an interleaved way. For instance, b_0 and b_1 of N will be stored in the one dimensional sequence as $b_0[0]b_1[0]b_0[1]b_1[1]$... instead storing all entries of b_0 together. In this case, we warrant consecutive threads will access adjacent data - when thread 0 accesses $b_0[0]$, thread 1 will be accessing $b_1[0]$ which is adjacent to $b_0[0]$.

Scalable H-INDEX. While scaling to multiple GPUs, the edge list was partitioned based upon the basis of their indices. H-INDEX simply assigns each GPU equal number of edges. At the end of the computation, H-INDEX relies on Message Passing Interface (MPI) to synchronize across all participating GPUs and arrive at the total number of triangles.

IV. EVALUATION

H-INDEX is implemented with around 500 lines of C++/CUDA code and compiled with CUDA Toolkit 10.1.105, g++ 6.4.0 and IBM Spectrum MPI 10.3.0.0 and the optimization flag set to -O3. We evaluate H-INDEX on V100 GPUs from the Summit Supercomputer [18], each node of which installs dual-socket 22-core POWER 9 processors and 512 GB

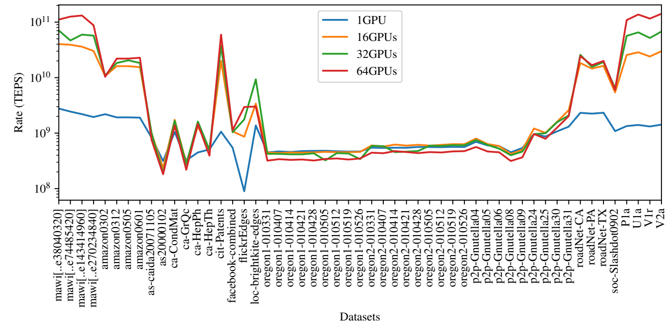


Fig. 2: TEPS for SNAP, MAWI and k-mer datasets.

memory. Particularly, each V100 GPU is equipped with 16GB device memory. For scalability test, we run H-INDEX with up to 64 GPUs that are distributed across multiple machines.

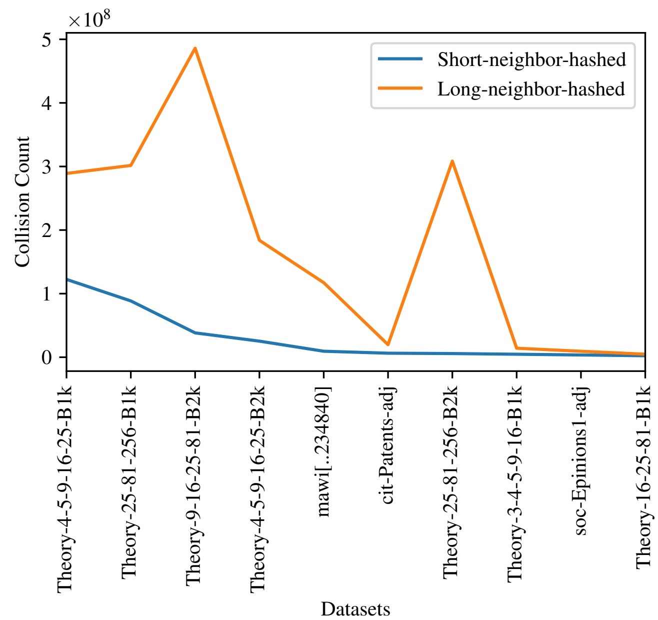


Fig. 3: Total collision count for datasets during hashing of neighbor list.

While computing TEPS, we only consider the kernel time or the time spent on GPU for counting triangles. This excludes the time for generating the CSR list, edge orientation and copying the graph to GPU. We mainly evaluate H-INDEX with the Stanford Network Analysis Project (SNAP) dataset [15], Protein K-mer dataset, MAWI dataset and Kronecker datasets mentioned in the Graph Challenge website.

Figure 2 plots the performance of SNAP, MAWI and K-mer datasets using 1, 16, 32 and 64 GPUs. The graph property of these datasets are listed in Table I, II and III, respectively. Impressively, majority of the MAWI and K-mer datasets show beyond 100 billion TEPS with 64 GPUs. On the contrary, H-INDEX only achieves 0.31 - 2.02 billion TEPS for a collection of oregen and p2p graphs with 64 GPUs.

Datasets	Vertices	Edges	Triangles	Rate (billion TEPS)			
				1GPU	**16GPUs**	**32GPUs**	**64GPUs**
amazon0302	262,111	899,792	717,719	2.194187	10.691221	10.970934	10.339729
amazon0312	400,727	2,349,869	3,686,467	1.921635	16.104681	18.353938	22.000145
amazon0505	410,236	2,439,437	3,951,063	1.923983	16.062387	20.50449	22.003743
amazon0601	403,394	2,443,408	3,986,507	1.899962	15.457611	18.366301	22.824935
as-caida20071105	26,475	53,381	36,365	0.835433	1.008541	1.008541	0.764151
as20000102	6,474	12,572	6,584	0.313874	0.233322	0.187654	0.18183
ca-CondMat	23,133	93,439	173,361	1.047892	1.734122	1.667709	1.37032
ca-GrQc	5,242	14,484	48,260	0.32314	0.289287	0.27365	0.219315
ca-HepPh	12,008	118,489	3,358,499	0.448537	1.332383	1.618824	1.461703
ca-HepTh	9,877	25,973	28,339	0.509059	0.509059	0.499719	0.39328
cit-Patents	3,774,768	16,518,947	7,515,023	1.061635	20.047884	37.71665	**59.421514**
facebook-combined	4,039	88,234	1,612,010	0.545037	1.075815	1.036639	1.131744
flickrEdges	105,938	2,316,948	107,987,357	0.089606	0.862288	1.763379	2.955591
loc-brightkite-edges	58,228	214,078	494,728	1.372948	3.401167	9.312991	3.013115
oregon1-010331	10,670	22,002	17,144	0.450161	0.447976	0.423317	0.319319
oregon1-010407	10,729	21,999	15,834	0.468378	0.439383	0.423259	0.33923
oregon1-010414	10,790	22,469	18,237	0.457485	0.448771	0.415162	0.329517
oregon1-010421	10,859	22,747	19,108	0.474666	0.445831	0.414817	0.334764
oregon1-010428	10,886	22,493	17,645	0.478896	0.44925	0.432764	0.320893
oregon1-010505	10,943	22,607	17,597	0.481323	0.460294	0.32362	0.342313
oregon1-010512	11,011	22,677	17,598	0.470863	0.452925	0.436304	0.349685
oregon1-010519	11,051	22,724	17,677	0.464933	0.453864	0.42933	0.334426
oregon1-010526	11,174	23,409	19,894	0.467545	0.458806	0.339739	0.349411
oregon2-010331	10,900	31,180	82,856	0.54719	0.599901	0.589092	0.444824
oregon2-010407	10,981	30,855	78,138	0.541486	0.572634	0.582952	0.434279
oregon2-010414	11,019	31,761	88,905	0.545964	0.622501	0.45466	0.474076
oregon2-010421	11,080	31,538	82,129	0.544362	0.595856	0.46414	0.456138
oregon2-010428	11,113	31,434	78,000	0.563435	0.616092	0.47597	0.436569
oregon2-010505	11,157	30,943	72,182	0.561837	0.595341	0.595341	0.455384
oregon2-010512	11,260	31,303	72,866	0.558699	0.613525	0.588764	0.448103
oregon2-010519	11,375	32,287	83,709	0.566617	0.632811	0.607271	0.468586
oregon2-010526	11,461	32,730	89,541	0.562621	0.629723	0.604756	0.473378
p2p-Gnutella04	10,876	39,994	934	0.698946	0.798795	0.752229	0.562909
p2p-Gnutella05	8,846	31,839	1,112	0.601543	0.635916	0.61258	0.466932
p2p-Gnutella06	8,717	31,525	1,142	0.585068	0.585068	0.524704	0.451281
p2p-Gnutella08	6,301	20,777	2,383	0.451529	0.423034	0.399748	0.314603
p2p-Gnutella09	8,114	26,013	2,354	0.540131	0.500488	0.464283	0.366129
p2p-Gnutella24	26,518	65,369	986	0.962026	1.213175	0.962026	0.962026
p2p-Gnutella25	22,687	54,705	806	0.869127	1.01079	0.993287	0.791205
p2p-Gnutella30	36,682	88,328	1,590	1.076961	1.603786	1.576487	1.260117
p2p-Gnutella31	62,586	147,892	2,024	1.308658	2.595414	2.081557	2.027137
roadNet-CA	1,965,206	2,766,607	120,676	2.321262	18.331739	25.844078	24.480993
roadNet-PA	1,088,092	1,541,898	67,150	2.244772	14.533009	15.583588	16.754376
roadNet-TX	1,379,917	1,921,660	82,869	2.329487	16.315843	19.054436	20.000065
soc-Slashdot0902	82,168	504,230	602,592	1.08679	5.422805	6.447847	6.008221

TABLE I: H-INDEX performance for SNAP dataset.

Datasets	Vertices	Edges	Triangles	Rate (billion TEPS)			
				1GPU	**16GPUs**	**32GPUs**	**64GPUs**
201512012345.v18571154-e38040320	18,571,154	38,040,320	2	2.774443	40.169353	70.692366	**111.264066**
201512020000.v35991342-e74485420	35,991,342	74,485,420	2	2.445993	39.120273	46.726668	**125.568529**
201512020030.v68863315-e143414960	68,863,315	143,414,960	6	2.197082	36.077847	59.580621	**131.567354**
201512020130.v128568730-e270234840	128,568,730	270,234,840	10	1.949604	30.230898	57.034523	87.768861

TABLE II: MAWI Dataset performance.

Datasets	Vertices	Edges	Triangles	Rate (billion TEPS)			
				1GPU	**16GPUs**	**32GPUs**	**64GPUs**
P1a	139,353,211	297,829,984	3412	1.340322	25.593425	56.524412	**108.890298**
U1a	67,716,231	138,778,562	325	1.404229	28.620291	65.402189	**137.02436**
V1r	214,005,017	465,410,904	49	1.318475	24.011351	51.827288	**115.774558**
V2a	55,042,369	117,217,600	1443	1.421991	29.918229	67.636023	**141.399555**

TABLE III: H-INDEX performance for k-mer dataset.

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

Datasets	Vertices	Edges	Triangles	Rate (billion TEPS)			
				1GPU	**16GPUs**	**32GPUs**	**64GPUs**
graph500-scale18-ef16	174,147	3,800,348	82,287,285	0.075299	0.96728	2.001232	**3.253687**
graph500-scale19-ef16	335,318	7,729,675	186,288,972	0.052944	0.705517	1.418535	2.573268
graph500-scale20-ef16	645,820	15,680,861	419,349,784	0.034848	0.450472	0.599995	1.145844
graph500-scale21-ef16	1,243,072	31,731,650	935,100,883	0.024424	0.330214	0.628051	1.21178
graph500-scale22-ef16	2,393,285	64,097,004	2,067,392,370	0.017704	0.242467	0.492248	0.868676
graph500-scale23-ef16	4,606,314	129,250,705	4,549,133,002	0.012377	0.166271	0.343543	0.619135
graph500-scale24-ef16	8,860,450	260,261,843	9,936,161,560	0.008746	0.111646	0.227191	0.439494
graph500-scale25-ef16	17,043,780	523,467,448	21,575,375,802	0.006335	0.082601	0.158073	0.296659

TABLE IV: H-INDEX performance for graph 500 dataset.

Datasets	Vertices	Edges	Triangles	Rate (billion TEPS)			
				1GPU	**16GPUs**	**32GPUs**	**64GPUs**
Theory-16-25-B1k	442	1,682	400	0.051152	0.034946	0.033614	0.025952
Theory-16-25-B2k	442	1,682	1	0.052679	0.034267	0.032986	0.025121
Theory-25-81-256-B1k	547,924	4,264,568	2,102,761	0.42518	1.940011	3.937244	7.166225
Theory-25-81-256-B2k	547,924	4,264,568	7	1.208493	5.265499	10.180364	**16.592671**
Theory-25-81-B1k	2,132	8,312	2,025	0.230909	0.173469	0.141163	0.159211
Theory-25-81-B2k	2,132	8,312	1	0.22495	0.176097	0.15094	0.118596
Theory-3-4-5-9-16-25-B1k	530,400	22,160,060	35,882,427	0.032511	0.341166	0.568512	1.050831
Theory-3-4-5-9-16-25-B2k	530,400	22,160,060	651	0.13769	1.558791	2.536321	3.468912
Theory-3-4-5-9-B1k	1,200	13,166	9,107	0.298521	0.264241	0.253332	0.190436
Theory-3-4-5-9-B2k	1,200	13,166	35	0.328729	0.262983	0.253332	0.187845
Theory-3-4-5-B1k	120	692	287	0.022359	0.014755	0.010531	0.009625
Theory-3-4-5-B2k	120	692	7	0.021691	0.014755	0.013333	0.010493
Theory-4-5-9-16-25-B1k	132,600	3,165,722	7,096,926	0.201662	1.94578	3.827617	6.692543
Theory-4-5-9-16-25-B2k	132,600	3,165,722	155	0.555961	4.516328	8.577522	11.813171
Theory-4-5-9-16-B1k	5,100	62,072	45,013	0.897769	1.14693	1.033147	0.910325
Theory-4-5-9-16-B2k	5,100	62,072	35	1.033147	1.13197	1.167502	0.888577
Theory-4-5-9-B1k	300	1,880	821	0.056759	0.037569	0.029438	0.027205
Theory-4-5-9-B2k	300	1,880	7	0.05717	0.039251	0.036025	0.026835
Theory-4-5-B1k	30	98	20	0.003099	0.001977	0.00194	0.001452
Theory-4-5-B2k	30	98	1	0.003099	0.00194	0.001432	0.001432
Theory-5-9-16-25-81-B1k	2,174,640	57,334,760	66,758,995	0.018054	0.216295	0.379597	0.662746
Theory-5-9-16-25-81-B2k	2,174,640	57,334,760	155	0.074344	0.714176	1.318801	2.107749
Theory-5-9-16-B1k	1,020	6,896	3,149	0.191577	0.146102	0.137753	0.104434
Theory-5-9-16-B2k	1,020	6,896	7	0.19679	0.146843	0.132698	0.102947
Theory-5-9-B1k	60	208	45	0.006743	0.004542	0.003949	0.003065
Theory-5-9-B2k	60	208	1	0.006542	0.004427	0.00373	0.003165
Theory-81-256-B1k	21,074	83,618	20,736	1.099447	1.49244	1.467462	1.146155
Theory-81-256-B2k	21,074	83,618	1	1.192937	1.572751	1.230609	1.192937
Theory-9-16-25-81-B1k	362,440	5,212,250	4,059,175	0.097144	1.159529	2.39214	3.531217
Theory-9-16-25-81-B2k	362,440	5,212,250	35	0.209749	1.876869	3.963337	4.18007
Theory-9-16-25-B1k	4,420	31,976	15,169	0.626735	0.638673	0.470601	0.484192
Theory-9-16-25-B2k	4,420	31,976	7	0.694929	0.651074	0.60415	0.468955
Theory-9-16-B1k	170	626	144	0.019626	0.013084	0.012063	0.009359
Theory-9-16-B2k	170	626	1	0.019626	0.013626	0.010822	0.009359

TABLE V: H-INDEX performance for Theory dataset.

Table IV and V further list out the performance for the Synthetic graphs, such as, graph 500 and Theory datasets. Note, in all the performance related tables, we highlight the TEPS of relatively better performance.

Figure 3 plots the hash collision count for various selected datasets like MAWI, SNAP and Theory graphs using two different approaches i.e hashing shorter vs. longer neighbor lists. As expected, hashing shorter neighbor list always introduces fewer collisions. Particularly, Theory-16-25-81-B1k dataset has a minimum hash collision of 2,304,450 and 4,758,124 with hashing shorter and longer neighbor lists, respectively. Conversely, Theory-4-5-9-16-25-B1k has a maximum hash collision of 122,394,672 and 289,068,158 with hashing shorter and longer neighbor lists, respectively.

Performance analysis. We find the performance is related to the following three factors. First, larger datasets like MAWI, road net have better performance than smaller datasets, e.g., p2p and oregon because they can better saturate the 64 GPUs. Second, a relatively more balanced distribution of the degree leads to better workload balance between the thread groups. Third, as H-INDEX uses vertex ID as a key to hash vertices into different bins, the ordering of vertex ID also has an impact on the performance.

Figure 4 and 5 plot a few datasets that present relatively better and worse scalability, respectively. The key reasons of such a dramatic difference lie in the vertex and edge count,

Datasets	Vertices	Edges	Triangles	Rate (billion TEPS)		
				[11] (2018 Champion)	[24] (2018 Champion)	H-INDEX
				8 x P100 GPU	Skylake CPU	1 x V100 GPU
Amazon0302	262,111	899,792	717,719	1.46	-	2.19
Amazon0312	400,727	2,349,869	3,686,467	2.64	0.387	1.922
roadNet-PA	1,088,092	1,541,898	67,150	1.73	-	2.245
roadNet-TX	1,379,917	1,921,660	82,869	2.03	-	2.33
soc-Slashdot0902	82,168	504,230	602,592	0.793	0.15	1.09

TABLE VI: H-INDEX vs. the champions from graph challenge 2018.

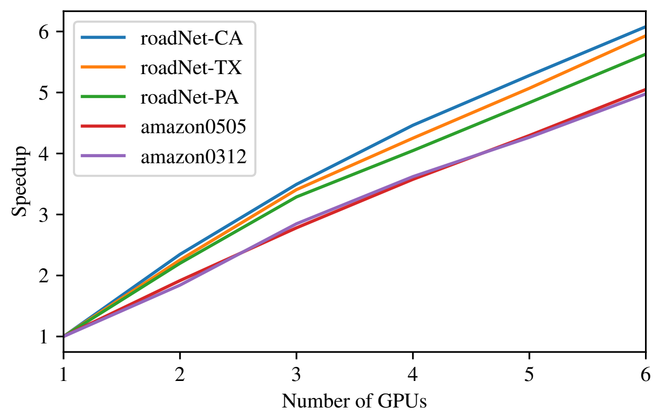


Fig. 4: Graphs with relatively better scalability.

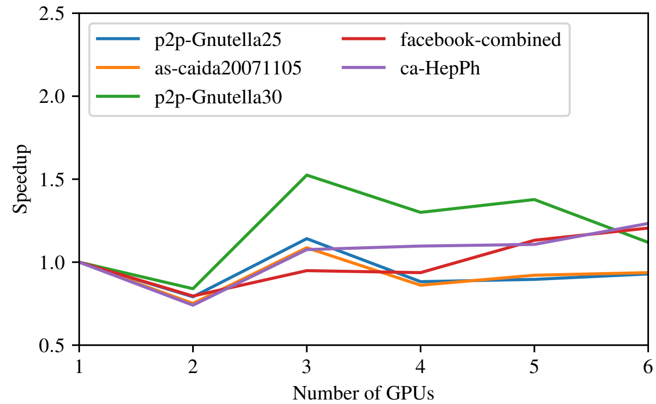


Fig. 5: Graphs with relatively worse scalability.

Dataset	Vertices	Edges	Triangles	Edge deg. stdev
roadNet-CA	1,965,206	2,766,607	120,676	1.12
roadNet-TX	1,379,917	1,921,660	82,869	1.13
roadNet-PA	1,088,092	1,541,898	67,150	1.14
amazon0505	410,236	2,439,437	3,951,063	3.93
amazon0312	400,727	2,349,869	3,686,467	3.85

TABLE VII: Dataset property for Figure 4.

Dataset	Vertices	Edges	Triangles	Edge deg. stdev
p2p-Gnutella25	22,687	54,705	806	3.16
as-caida20071105	26,475	53,381	36,365	8.78
p2p-Gnutella30	36,682	88,328	1,590	3.45
facebook_combined	4,039	88,234	1,612,010	51.38
ca-HepPh	12,008	118,489	3,358,499	99.92

TABLE VIII: Dataset property for Figure 5.

and the edge degree distribution. Particularly, as shown in Table VII and VIII, first, graphs with more vertices and edges can better saturate the GPU computing resources. Second, graphs with balanced edge degrees lead to balanced workload distributions.

Table VI compares the performance of our implementation with the champions in graph challenge 2018. [11] is a GPU based implementation while [24] is a CPU based one. The performance of [11] is achieved with 8× P100 GPUs, which is taken from the paper. And [24] uses a 24-core Intel Xenon Platinum 8160 processor with 33MB L3 cache. According to Amazon, one V100, 8× P100 and 24-core Intel Xeon Platinum 8160 processor cost 4,979 USD [2], 32,472 USD [3] and 4,237 USD [1], respectively.

V. Conclusion

This paper proposes and implements the H-INDEX based approach for triangle counting that avoids the preprocessing step of sorting the neighbor list. For better memory access pattern, we further introduce interleaved format for the hash bucket storage. Taken together, H-INDEX achieves beyond 100 billion TEPS computing rate for some graphs.

Acknowledgment

We thank the anonymous reviewers for their constructive suggestions that help improve the quality of this paper. We also would like to gracefully acknowledge the support from XSEDE supercomputers and Amazon AWS, as well as the NVIDIA Corporation for the donation of the Titan Xp and Quadro P6000 GPUs. This work was in part supported by National Science Foundation (NSF) CRII Award No. 1850274, the Sustainable Research Pathway (SRP) Fellowship of the U.S. Department of Energy (DOE), and 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 (NNSA). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DOE, NNSA or NSF.

REFERENCES

[1] AMAZON. Price of Intel Xeon Platinum 8160 Processor. Retrived from https://www.amazon.com/dp/B07V3SYQ1P/ref=nav_timeline_asin?_encoding=UTF8&psc=1. Accessed: 2019, August 28.

[2] AMAZON. Price of Tesla P100 12 GB. Retrived from https://www.amazon.com/dp/B01KVHKMOK/ref=nav_timeline_asin?_encoding=UTF8&psc=1. Accessed: 2019, August 28.

[3] AMAZON. Price of Tesla V100 16 GB. Retrived from https://www.amazon.com/dp/B076P84525/ref=nav_timeline_asin?_encoding=UTF8&psc=1. Accessed: 2019, August 28.

[4] AZAD, A., BULUÇ, A., AND GILBERT, J. R. Parallel triangle counting and enumeration using matrix algebra. In *Proceedings of the IPDPSW, Workshop on Graph Algorithm Building Blocks (GABB)* (2015), pp. 804 – 811.

[5] BHATTARAI, B., LIU, H., AND HUANG, H. H. Ceci: Compact embedding cluster index for scalable subgraph matching. In *Proceedings of the 2019 International Conference on Management of Data* (2019), ACM, pp. 1447–1462.

[6] BISSON, M., AND FATICA, M. High performance exact triangle counting on gpus. *IEEE Transactions on Parallel and Distributed Systems 28*, 12 (2017), 3501–3510.

[7] GAIHRE, A., WU, Z., YAO, F., AND LIU, H. Xbfs: exploring runtime optimizations for breadth-first search on gpus. In *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing* (2019), ACM, pp. 121–131.

[8] GIECHASKIEL, I., PANAGOPOULOS, G., AND YONEKI, E. Pdtl: Parallel and distributed triangle listing for massive graphs. In *2015 44th International Conference on Parallel Processing* (2015), IEEE, pp. 370–379.

[9] GREEN, O., YALAMANCHILI, P., AND MUNGUÍA, L.-M. Fast triangle counting on the gpu. In *Proceedings of the 4th Workshop on Irregular Applications: Architectures and Algorithms* (2014), IEEE Press, pp. 1–8.

[10] HU, Y., KUMAR, P., SWOPE, G., AND HUANG, H. H. Trix: Triangle counting at extreme scale. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)* (2017), IEEE, pp. 1–7.

[11] HU, Y., LIU, H., AND HUANG, H. H. High-performance triangle counting on gpus. In *2018 IEEE High Performance extreme Computing Conference (HPEC)* (2018), IEEE, pp. 1–5.

[12] HU, Y., LIU, H., AND HUANG, H. H. Tricore: Parallel triangle counting on gpus. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis* (2018), IEEE, pp. 171–182.

[21] SESHADHRI, C., PINAR, A., AND KOLDA, T. G. Wedge sampling for computing clustering coefficients and triangle counts on large graphs. *Statistical Analysis and Data Mining: The ASA Data Science Journal 7*, 4 (2014), 294–307.

[13] JI, Y., LIU, H., AND HUANG, H. H. ispan: Parallel identification of strongly connected components with spanning trees. In *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis* (2018), IEEE, pp. 731–742.

[14] LATAPY, M. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theoretical computer science 407*, 1-3 (2008), 458–473.

[15] LESKOVEC, J., AND SOSIČ, R. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST) 8*, 1 (2016), 1.

[16] LIU, H., AND HUANG, H. H. Enterprise: breadth-first graph traversal on gpus. In *SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2015), IEEE, pp. 1–12.

[17] LOW, T. M., RAO, V. N., LEE, M., POPOVICI, D.-T., FRANCHETTI, F., AND MCMILLAN, S. First look: Linear algebra-based triangle counting without matrix multiplication. *2017 IEEE High Performance Extreme Computing Conference (HPEC)* (2017), 1–6.

[18] OAK RIDGE NATIONAL LAB. SUMMIT Oak Ridge National Laboratory's 200 petaflop supercomputer. Retrived from https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/. Accessed: 2019, July 6.

[19] PEARCE, R., AND SANDERS, G. K-truss decomposition for scale-free graphs at scale in distributed memory. In *2018 IEEE High Performance extreme Computing Conference (HPEC)* (2018), IEEE, pp. 1–6.

[20] SCHANK, T., AND WAGNER, D. *Approximating clustering-coefficient and transitivity*. Universität Karlsruhe, Fakultät für Informatik, 2004.

[22] SHUN, J., AND TANGWONGSAN, K. Multicore triangle computations without tuning. In *2015 IEEE 31st International Conference on Data Engineering* (2015), IEEE, pp. 149–160.

[23] WANG, L., WANG, Y., YANG, C., AND OWENS, J. D. A comparative study on exact triangle counting algorithms on the gpu. In *Proceedings of the ACM Workshop on High Performance Graph Processing* (2016), ACM, pp. 1–8.

[24] YASAR, A., RAJAMANICKAM, S., WOLF, M. M., BERRY, J. W., AND ÇATALYÜREK, Ü. V. Fast triangle counting using cilk. *2018 IEEE High Performance extreme Computing Conference (HPEC)* (2018), 1–7.

[25] ZHANG, J., SPAMPINATO, D. G., MCMILLAN, S., AND FRANCHETTI, F. Preliminary exploration of large-scale triangle counting on shared-memory multicore system. In *2018 IEEE High Performance extreme Computing Conference (HPEC)* (2018), IEEE, pp. 1–6.