View Reviews

Paper ID
515

Paper Title
Dynamic Hash Tables on GPUs

Track Name
Research -> January 2019

Reviewer #1

Questions

1. Overall Rating
Weak Reject

2. Relevant for PVLDB
Yes

3. Are there specific revisions that could raise your overall rating?
No

4. Summary of the paper (what is being proposed and in what context) and a brief justification of your overall recommendation. One paragraph
The paper proposes a technique for building dynamic hash tables on GPUs. The novelty is a coordination strategy for resolving thread conflicts in a GPU. Experiments validate the effectiveness of the proposed design vs. current GPU hash table implementations.

5. Three (or more) strong points about the paper (Please be precise and explicit; clearly explain the value and nature of the contribution).
S1. The paper solves an interesting problem in the context of GPU based hashing
S2. The use of existing techniques such as Cuckoo hashing makes sense
S3. Presentation is good

6. Three (or more) weak points about the paper (Please indicate clearly whether the paper has any mistakes, missing related work, or results that cannot be considered a contribution; write it so that the authors can understand what is seen as negative)
W1. The main novelty of the paper comes from a novel coordination strategy and the dynamic resizing aspect, but it is not clear how important either of these contributions are in reality.
W2. The paper does not evaluate against skewed data distributions for inserts and updates, which are necessary to understand the effect of e.g., the coordination techniques.
W3. Comparison against current state of the art for dynamic hashing on GPU is missing.

7. Novelty (Please give a high novelty ranking to papers on new topics, opening new fields, or proposing truly new ideas; assign medium ratings to delta papers and papers on well-known topics but still with some valuable contribution).
With some new ideas

8. Significance
Improvement over existing work

9. Technical Depth and Quality of Content
Solid work

11. Presentation

Reasonable: improvements needed

12. Detailed Evaluation (Contribution, Pros/Cons, Errors); please number each point

D1. The impact of dynamic resizing and the design on average latency and worst case latency would be useful to better understand the downsides of dynamic resizing.
D2. The fixed size hash tables such as MegaKV show clear superiority in several scenarios. Can the design be made adaptive so that the penalty when it is not necessary to resize is not incurred?
D3. The paper claims the coordination strategy to be a novel contribution. However, the superiority of the proposed technique over traditional coordination is not well explained, analyzed, or evaluated in the paper. It seems that the authors' solution does not really eliminate the coordination overhead shown in Fig. 3, rather it reduces it by a constant factor.
D4. The dynamic resizing aspect is motivated in the introduction, but it's need should be better argued using real world examples. How often are hash tables used inside a GPU, and which of these scenarios would benefit from resizing?
D5. Cuckoo hashing is generally optimized for reads, but does poorly on updates. Is this true in the GPU setting as well?
D6. The authors use the term "warp" in the introduction without explaining it. In general, the paper would benefit from a better introduction to programming GPUs, as the current description makes it hard to follow the parallel nature of the algorithm. For example, Algorithm 1 takes warp id (wid) as input, but this is not referenced in the algorithm at all.
D7. The paper does not evaluate against skewed data distributions for inserts and updates, which are necessary to understand the effect of e.g., the coordination techniques. Can they use Zipf distributions with factor varying from say 0.75 to 3, and demonstrate the effect on throughput, avg. and worst case latency?
D8. Given that the only dynamic hash table for GPU is [4] it might be worthwhile to implement that technique based on details from the paper, and use it as a comparison basis. Static tables are not a good baseline given that the key contribution of the work is its dynamic nature.
D9. A graph of throughput vs. time as the hash table changes size, would be a useful addition to better understand the runtime characteristics of the data structure.

Reviewer #2

Questions

1. Overall Rating

Weak Reject

2. Relevant for PVLDB

Yes

3. Are there specific revisions that could raise your overall rating?

Yes

4. Summary of the paper (what is being proposed and in what context) and a brief justification of your overall recommendation. One paragraph

This paper solves the problem of efficiently supporting dynamic hash tables in GPU with two main techniques. The first is a voting mechanism that coordinates the threads in one warp to resolve conflicts and fully utilize parallelism. The second is a resizing strategy that adjusts one hash table at a time and adjusts only part of the entries by rehashing. The experiments show that the proposed techniques (DyHash) are significantly more efficient than existing basslines under dynamic hash table resizing.

5. Three (or more) strong points about the paper (Please be precise and explicit; clearly explain the value and nature of the contribution).

S1. Section 2.2 presents a nice introduction of GPU architecture and design guidelines of efficient GPU algorithms.
S2. The voting mechanism is nice and presented clearly with examples.
S3. The KV distribution strategy in Section 5.2 is backed up by theory.
S4. Extensive experiments were conducted.

6. Three (or more) weak points about the paper (Please indicate clearly whether the paper has any mistakes, missing related work, or results that cannot be considered a contribution; write it so that the authors can understand what is seen as negative)
W1. The motivations are not well justified.
W2. The approaches are simple and straightforward.
W3. The experiment section is hard to follow.
W4. The proposed method does not perform well under static hashing.

7. Novelty (Please give a high novelty ranking to papers on new topics, opening new fields, or proposing truly new ideas; assign medium ratings to delta papers and papers on well-known topics but still with some valuable contribution).
With some new ideas

8. Significance
Improvement over existing work

9. Technical Depth and Quality of Content
Syntactically complete but with limited contribution

11. Presentation
Sub-standard: would require heavy rewrite

12. Detailed Evaluation (Contribution, Pros/Cons, Errors); please number each point
D1. Two important motivations of the paper include: (1) existing approaches support only up to 64 bits for KV altogether; (2) without dynamic resizing, existing approaches waste memory, which makes it difficult for other applications to coexist with the hash table on GPU memory. The first claim should be justified by examples in which 64-bit KV pair is not sufficient. The second claim should be justified by examples by use cases that run other applications with the hash tables. Moreover, claim 2 (the memory saving of DyHash) should be backed up experiments comparing the memory consumptions of the baselines.

D2. The techniques are straightforward and the contributions are questionable. (1) Adjusting the number of KV pairs in one bucket to adapt to different sizes of the KV pair is straightforward. (2) The novelty of using a voting mechanism to resolve conflicts within a warp is problematic. (3) Resizing only one hash table when needed and rehashing only part of the entries (the other entries are processed using cheaper heuristics) seem to be incremental. I am not an expert in GPU and it’s hard for me to tell (2) is novel or just routine conflict handling in GPU.

D3. In the experiment section, DyHash does not perform well for static hashing. Under dynamic hashing, the main performance gain of DyHash comes from its more efficient resizing. If resizing is not required, Linear seems to be the best choice. It should be made clear the scenario in which DyHash is most suitable.

D4. As stated in D1, memory composition of the baselines should be tested in the experiments.

D5. Using larger datasets (e.g. billion scale) and datasets with much larger max duplicates (currently 14) will make the experiments more convincing.

Some minor points:
M1. In section 4.2, it is assumed that find, insert and delete operations are batched and each batch only contains one type of operations. Can you give the justifications for the assumptions?

M2. The second paragraph of Section 5.2 is hard to follow. How can we get the formulas and filled factor relations? Figure 8 should be Figure 5.

M3. The experiments are rather lengthy and there are many figures and tables. Please try to summarize the main results in each part.

13. If revision is required, list specific revisions you seek from the authors
Please address D1 to D5 in Detailed Evaluation.

Reviewer #3

Questions

1. Overall Rating
Reject

2. Relevant for PVLDB
Yes

3. Are there specific revisions that could raise your overall rating?
No

4. Summary of the paper (what is being proposed and in what context) and a brief justification of your overall recommendation. One paragraph
The authors propose two innovations for cuckoo hash table on GPUs. The first is a voter-based conflict resolution scheme in case multiple threads want to update the same key. The second innovation is an improved resizing strategy which only resizes a single subtable and reduces conflicts during hashing.

5. Three (or more) strong points about the paper (Please be precise and explicit; clearly explain the value and nature of the contribution).
S1: Interesting new concept on voter-based cuckoo hashing
S2: Includes proofs of dynamic resizing costs
S3: Evaluation is thorough and includes real-world data sets

6. Three (or more) weak points about the paper (Please indicate clearly whether the paper has any mistakes, missing related work, or results that cannot be considered a contribution; write it so that the authors can understand what is seen as negative)
W1: Not distinguish to related work on CPU-based dynamic hash tables
W2: Weak use of English language
W3: The provided pseudocode and math is not comprehensible
W4: This paper does not evaluate the dynamic resizing without the voter-based conflict resolution.

7. Novelty (Please give a high novelty ranking to papers on new topics, opening new fields, or proposing truly new ideas; assign medium ratings to delta papers and papers on well-known topics but still with some valuable contribution).
With some new ideas

8. Significance
No impact

9. Technical Depth and Quality of Content
Insignificant contribution

11. Presentation
Sub-standard; would require heavy rewrite

12: Detailed Evaluation (Contribution, Pros/Cons, Errors), please number each point

D1: The language is not on VLDB standard. For example on first page: “takes unnecessarily large memory resources”,
“makes rooms”, “egoism renders inefficiencey”, “Imaging . . . ”; Page 2: “otherwise to trigger”, “seminar”, “delete, O(1)” (should be “and”); Page 5: “In the meanwhile”, Page 7: “be as twice large as”, Page 7: “Similar as”, “resolve race condition”
D2: Table 1 is not referenced in text.
D3: Footnote 2 is missing a reference.
D4: 4.1: The second point in the first paragraph implies that hash tables are stored interleaved for coalesced access in all hash tables (“. . . one only needs to search d locations . . . the data could be stored contiguously in the same location and thus enabled the preferred coalesced memory access.”). This contradicts Figure 2. Also, because each hash table has its own hashing function, the hash values would not be the same.
D5: 4.1: If a bucket is always 128 bytes (cache line) then not all threads of a warp are utilized if keys are bigger than 4 bytes? What happens when keys are smaller than 4 bytes? Is the hash bucket padded to 128 bytes? Or do threads access multiple keys in the bucket?
D6: 4.2: I do not understand the discussion in the beginning of the insert section. Why should an entire warp perform an insert instead of a single thread?
D7: Complexity analysis: How does m/2 grows compared to H? Short discussion or example would be helpful. Also, the formula (m/2) /H does not appear in source 23.
D8: 5.1 Figure 8 probably refers to Figure 5.
D9: Definition of smallest and largest hash table is not clear. Shouldn’t there be just 2 different sizes for all hash tables, if we start with a reasonable assumption that all tables have the same size initially?
D10: Why is insertion complexity O(1) important for the proof?
D11: 5.3: It would be helpful to repeat the hashing function to make clear why a key stays in the bucket or goes to bucket loc + |h i |.
D12: Page 7, 2nd sentence: “Does not require locking”.
D13: 5.3/Complexity: Downsizing rehashes more than m/d KV pairs if large tables have higher fill factor? Also, why is the number of rehashes bounded by 2m?
D14: Why does downsize have a smaller throughput than upsize in Figure 8? If it’s completely lock-free, shouldn’t it be faster? Downsize has to insert some keys in the other table which can lead to subsequent evictions.
D15: Competitive in dynamic setting? Up to 2× slower.
D16 It would be interesting to see how MegaKV performs with the proposed resizing strategy. It has more failures but has 2× faster insertions. If resizes are cheaper, maybe overall performance would be better?
D17 Titles in references are not capitalized properly (e.g., 15) or miss punctuation (e.g., 23).
D18: Authors claim that their algorithm has benefits for multi-tasking, but never show such a workload in the evaluation.
D19: Unclear how multiple SMs coordinate work. It appears as if the authors advertise a lock-free algorithm. However, in Section 4.2 they clarify that locks are indeed necessary, but only acquired once per bucket instead of on each entry.
D20: Algorithm 1&2 ballot and broadcast functions are unclear, as there are more outputs than inputs to the function. In practice, CUDA provides _ shflsync (), which takes a variable "T" as input and gives a (possibly different) T as output. Perhaps the authors could adapt

ballot and broadcast to clarify their inputs.
D21: Algorithm 2: On line 22, i <- i + 1 should probobaly read: i <- (i + 1) mod d
D22: The TPC-H Lineitems data set is generated using a uniform ran-
dom distribution. How does it differ from the authors' RAND
data set?
D23: The Twitter and Reddit data sets are not publicly available. Per-
haps the authors could provide them on their website?
D24: It's unclear if and how skewed any of these data sets are. Perhaps
the authors could clarify, or provide a Zipf distribution instead?