GPU Hashing Data Structures and their Application in Accelerated Genomics

Daniel Jünger and Bertil Schmidt

Institute of Computer Science
Johannes Gutenberg-University, Mainz, Germany

NHR PerfLab Seminar
It is all about memory bandwidth!

**DDR4 modules built in Xeon multi-socket workstations**

- Few hundred GB/s
- a few TB of size

**HBM2 stacked memory modules attached to Tesla P100/V100/A100**

- up to 2 TB/s
- less than 80 GB (A100)
Why hashing is a good idea

Hash tables are well-suited if range queries do not matter:

<table>
<thead>
<tr>
<th></th>
<th>Hash Table</th>
<th>Sorted Array</th>
<th>Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>insertion per element</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>query per element</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>peak memory</td>
<td>$(1 + \varepsilon)n$</td>
<td>$2n$</td>
<td>$(1 + \varepsilon)n$</td>
</tr>
<tr>
<td>final memory</td>
<td>$(1 + \varepsilon)n$</td>
<td>$n$</td>
<td>$(1 + \varepsilon)n$</td>
</tr>
<tr>
<td>range queries</td>
<td>not supported</td>
<td>supported</td>
<td>supported</td>
</tr>
</tbody>
</table>

- out-of-place sorting usually needs $O(n)$ auxillary memory: CUDA Unbound radix sort uses double buffers → waste of valuable video memory
- incomplete trees exhibit highly irregular data layouts and are hard to construct in parallel without auxillary memory
We propose WarpCore - a versatile library of hashing data structures

• **Performance**
  • main focus on high-throughput table operations
  • WarpCore outperforms other state-of-the-art CPU and GPU hash tables

• **Modularity**
  • building blocks for constructing customized GPU hash tables
  • probing schemes, hashers, memory layouts, etc.

• **Host-sided and device-sided interfaces**
  • host-sided (bulk) operations provide high throughput
  • device-sided operations (fuse table operations with other tasks in one kernel)

• **Fully-asynchronous execution**
  • allows for task overlapping and multi-GPU setups

• Jünger, Kobus, Müller, Hundt, Xu, Liu, Schmidt: “WarpCore: A Library for fast Hash Tables on GPUs”, IEEE HiPC 2020
Parallel Hash Table Construction

Scenario: inserting new key/value pairs into a hash table in parallel

- determine slot index for $k_A$ by applying a hash function $h(k_A) \mod c = 6$
- write $(k_A, v_A)$ to the target slot
- subsequent retrieval of the same element works in the same fashion

- hash collisions among keys
  - $h(k) \mod c = h(k') \mod c$ for $k \neq k'$
  - for suitable resolution strategies see next slide

- race conditions in a parallel setup
  - can be avoided by using atomic operations (CAS)
<table>
<thead>
<tr>
<th>Collision Resolution Strategies</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Separate Chaining</strong></td>
<td><strong>Open Addressing</strong></td>
</tr>
<tr>
<td><em>Slots (buckets) store multiple colliding key-value pairs.</em></td>
<td><em>Find the next unoccupied slot by means of a deterministic probing scheme.</em></td>
</tr>
<tr>
<td>• Dynamic Linked Lists</td>
<td>• Linear Probing: $s(k, i) = (h(k) + i \mod m$</td>
</tr>
<tr>
<td>• allows for dynamic table growth</td>
<td>• cache efficient</td>
</tr>
<tr>
<td>• overhead due to memory allocations</td>
<td>• prone to primary and secondary clustering</td>
</tr>
<tr>
<td>• slow pointer chasing during bucket iteration</td>
<td>• Quadratic Probing: $s(k, i) = (h(k) + i^2 \mod m$</td>
</tr>
<tr>
<td>• Static Arrays</td>
<td>• leaves dense regions faster than linear probing</td>
</tr>
<tr>
<td>• memory over-provisioning</td>
<td>• prone to secondary clustering, i.e., $s(k, 0) = s(k', 0)$</td>
</tr>
<tr>
<td>• requires additional array iteration during probing</td>
<td>• Double Hashing: $s(k, i) = (h_1(k) + i \times h_2(k)) \mod m$</td>
</tr>
<tr>
<td></td>
<td>• if $m$ is prime and $0 &lt; h_2(k) &lt; m$ then</td>
</tr>
<tr>
<td></td>
<td>• $s(k, 0) \neq s(k', 0)$, i.e., no secondary clustering</td>
</tr>
<tr>
<td></td>
<td>• $s(k, i)$ for $i &lt; m$ is cycle-free</td>
</tr>
<tr>
<td></td>
<td>• Cuckoo Hashing</td>
</tr>
<tr>
<td></td>
<td>• greedily swap keys between candidate positions</td>
</tr>
<tr>
<td></td>
<td>• may result in infinite cycles</td>
</tr>
<tr>
<td></td>
<td>• Robinhood Hashing</td>
</tr>
<tr>
<td></td>
<td>• takes from the rich and gives to the poor</td>
</tr>
<tr>
<td></td>
<td>• reduces probing length variance</td>
</tr>
</tbody>
</table>
Cooperative Probing Scheme

- exploits fast intra-warp communication via registers
- intra-group linear probing + inter-group chaotic probing

Considerations for multi-value scenarios:
- probing scheme has to be cycle-free (e.g. double hashing)
- retrieval can be done cooperatively
- storing identical keys multiple times is memory inefficient
Open addressing hash tables lack space efficiency for highly skewed data.

Alternative approach:

- store keys only once in a single-value OA hash table
- each key holds a handle to a list of values
- each list consists of linked buckets of varying size
- buckets reside inside a pre-allocated memory pool
Bucket List Hash Table

- **hash table**:
  - keys
  - handles

- **value lists**:
  - values

- **in parallel**:
  - retrieve($k_1$):
  - retrieve($k_2$):
  - retrieve($k_3$):

- **COPS**

- **hashing coalesced access**

- **free slots**

- **bucket header**
  - index of previous bucket
  - bucket list handle
  - total value count

- **indexes**:
  - index of last bucket
  - bucket list handle

- **keys, handles, value lists**
  - $k_1$: [e, e, p]
  - $k_2$: [o, n]
  - $k_3$: [h, a, s, h, i, n, g]
Single-GPU Single-Value Performance

Bulk performance 4+4 byte and (U32) and 8+8 byte (U64) key-value pairs

(a) Tesla V100 (System 1)
**Single-GPU Multi-Value Performance**

**Bulk performance with average key multiplicity of 8**

---

**INSERTION**

- **U32**
- **U64**

**RETRIEVAL**

- **U32**
- **U64**

Operations per second

- WC CG=4
- WC CG=8
- CUDF
- WC BLHT

---

(a) Tesla V100 (System 1)

---

GPU Hashing Data Structures and their Application in Accelerated Genomics
### Multi-GPU Hashing

#### Random Keys

<table>
<thead>
<tr>
<th>GPU 0</th>
<th>GPU 1</th>
<th>GPU 2</th>
<th>GPU 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>12</td>
<td>19</td>
<td>50</td>
</tr>
<tr>
<td>40</td>
<td>67</td>
<td>20</td>
<td>11</td>
</tr>
<tr>
<td>35</td>
<td>14</td>
<td>37</td>
<td>51</td>
</tr>
<tr>
<td>44</td>
<td>94</td>
<td>74</td>
<td>42</td>
</tr>
<tr>
<td>56</td>
<td>86</td>
<td>57</td>
<td>31</td>
</tr>
<tr>
<td>16</td>
<td>89</td>
<td>73</td>
<td>62</td>
</tr>
<tr>
<td>26</td>
<td>53</td>
<td>25</td>
<td>43</td>
</tr>
</tbody>
</table>

#### Multi-split with $p(k) = k \% 4$

<table>
<thead>
<tr>
<th>GPU 0</th>
<th>GPU 1</th>
<th>GPU 2</th>
<th>GPU 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>12</td>
<td>20</td>
<td>50</td>
</tr>
<tr>
<td>44</td>
<td>89</td>
<td>37</td>
<td>42</td>
</tr>
<tr>
<td>56</td>
<td>53</td>
<td>57</td>
<td>62</td>
</tr>
<tr>
<td>16</td>
<td>14</td>
<td>73</td>
<td>11</td>
</tr>
<tr>
<td>17</td>
<td>94</td>
<td>25</td>
<td>51</td>
</tr>
<tr>
<td>26</td>
<td>86</td>
<td>74</td>
<td>31</td>
</tr>
<tr>
<td>20</td>
<td>73</td>
<td>19</td>
<td>43</td>
</tr>
</tbody>
</table>

#### All-to-All

<table>
<thead>
<tr>
<th>GPU 0</th>
<th>GPU 1</th>
<th>GPU 2</th>
<th>GPU 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>17</td>
<td>26</td>
<td>35</td>
</tr>
<tr>
<td>44</td>
<td>89</td>
<td>19</td>
<td>62</td>
</tr>
<tr>
<td>56</td>
<td>53</td>
<td>25</td>
<td>43</td>
</tr>
<tr>
<td>16</td>
<td>94</td>
<td>74</td>
<td>31</td>
</tr>
<tr>
<td>12</td>
<td>73</td>
<td>51</td>
<td>43</td>
</tr>
<tr>
<td>20</td>
<td>86</td>
<td>19</td>
<td>43</td>
</tr>
<tr>
<td>20</td>
<td>73</td>
<td>25</td>
<td>43</td>
</tr>
</tbody>
</table>

- Gossip communication library: 1.8 TB/s (0.5 TB/s) on DGX-2 (DGX-1) for All-to-All
  - Jünger, Hundt, Schmidt: *WarpDrive: Massively Parallel Hashing on Multi-GPU Nodes*, IPDPS 2018
  - Kobus, Jünger, Hundt, Schmidt: *Gossip: Efficient Communication Primitives for Multi-GPU Systems*, ICPP 2019
Multi-GPU Single-Value Performance

Weak scalability analysis on a DGX-1 server with 2GB of key-value pairs per GPU.

WarpCore achieves 100.8 GB/s throughput using 8 Tesla V100 at a scaling efficiency of 53%.
Next-Generation Sequencing (NGS)

DNA → [Next-Generation Sequencing (NGS)] → [Read-sequences] → [Genome sequence]

May contain errors!

**Illumina NovaSeq 6000 (Towards $100 per Genome)**
- **Read length**: 2\(\times\)150 bps
- **Reads per run**: 20 billion
- **Run Time**: <2 days

**Cost per Genome**
- Recorded growth
- Double every 7 months (historical growth rate)
- Double every 12 months (Illumina estimate)
- Double every 18 months (Moore's law)

**Graphs**
- **Correlation of Number of Human Genomes**
- **Worldwide Human Genomes Sequencing Capacity**
• Genomic sequences obtained directly from an environment (e.g. soil, gums, food, air, …)
• Reads stem from a mix of genomes ⇒ taxonomic read assignment problem
• NGS generates vast amounts of data ⇒ data set sizes and reference genome databases are increasing rapidly
Collaboration with Prof. T. Hankeln’s group (Biology, JGU)
Kraken: Taxonomic Classification of Reads

- each $k$-mer of input read mapped to the LCA of the genomes that contain that $k$-mer using a (pre-computed) $k$-mer index
- **Advantages**: Orders-of-magnitude faster than alignment, relatively simple
- **Disadvantages**: Huge $k$-mer index, random lookups,
MinHashing

- **Minhashing** can be used to estimate **Jaccard similarity** of two sets:
  \[ \Pr(h_{\min}(A) = h_{\min}(B)) = J(A, B) \]

- Apply hash function \( h \) to all \( k \)-mers and sketch

  \[ S_s(X) = \text{set of } s \text{ smallest hash values } h(x) \text{ of all } x \in X \]

### 3-mer hashes:

<table>
<thead>
<tr>
<th>CTAGC TTAATAT</th>
<th>( A_h = { 83, 229, 55, 198, 128, 184, 79, 57, 188, 165 } )</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTAGC A TAATAT</td>
<td>( B_h = { 83, 229, 55, 81, 90, 188, 79, 57, 188, 165 } )</td>
</tr>
</tbody>
</table>

\[
M_s(A, B) = \frac{|S_s(A) \cap S_s(B)|}{|S_s(A) \cup S_s(B)|} \approx J(A, B)
\]

- \( S_4(A_h) = \{55, 57, 79, 83\} \)
- \( S_4(B_h) = \{55, 57, 79, 81\} \)
- \( M_4(A, B) = \frac{3}{5} = 0.6 \)
MetaCache-GPU

Build Phase

Query Phase

...  

hashes of k-shingles  

min hashing signature  

CANDIDATES

window range  

hits  

CLASSIFICATION

HASH MAP

key  
genome locations  

accumulate & sort hits
Tested on sequenced “calibrator sausages”
Comparison of metagenomic database construction times for **151 GB** of genomes using a custom WarpCore hash table on 8 GPUs.  
⇒enables “On-the-Fly” metagenomics

- **Kobus, Müller, Jünger, Hundt, Schmidt:** *MetaCache-GPU: Ultra-Fast Metagenomic Classification*, ICPP 2021
\( k \)-mer Counting

\[ R = \{ \text{ACGTTA, ACGTTA, ACGTTT} \} \]

\( k = 4 \)

- \( k \)-mer counting is required by many bioinformatics tools; e.g. genome assembly, error correction, multiple sequence alignment, repeat detection

<table>
<thead>
<tr>
<th>( k )-mer</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACGT</td>
<td>3</td>
</tr>
<tr>
<td>CGTT</td>
<td>3</td>
</tr>
<tr>
<td>GTTA</td>
<td>2</td>
</tr>
<tr>
<td>GTTT</td>
<td>1</td>
</tr>
</tbody>
</table>
• **k-mer abundance histograms** could be used as indicator for contamination

• For haploid organisms: the distribution should resemble a single Gaussian + low frequency k-mers indicating sequencing errors
Parallel Hashing on Multi-GPU Nodes

WarpCount

WarpCore

GPU

CPU

parse

parse

thread 0

thread 1

encode

encode

kmerize

kmerize

kmerize

all 2

all

bloom filters

hash tables

filter

filter

filter

count

count

count

retrieve all

decode

decode

write result

counts

ACGT,3

GTCC,4

CCCT,2

...
WarpCount: Performance Evaluation

- WarpCore outperforms the fastest CPU-based k-mer counting tool (KMC 3) by a factor of up to 13x on a single V100
- The multi-GPU setup alleviates GPU memory limitations and thus makes processing of large datasets possible
We have presented WarpCore - a versatile library of GPU hash table data structures.

• a framework for high-throughput hashing-based data structures that can be tailored to fit many use cases
• efficient implementations of single- and multi-value hash tables, hash sets, counting hash tables, and bloom filters
• we propose a new multi-value hash table approach which provides robust throughput at high memory densities even for highly skewed input distributions
• easily scalable over up-to 16 GPUs (DGX-2)
• Can be used for a variety of applications in bioinformatics (e.g. metagenomics, k-mer counting)
Thank You!

• Daniel Jünger, Robin Kobus, André Müller, Bertil Schmidt
  • {juenger, kobus, muellan, bertil.schmidt}@uni-mainz.de
  • Johannes Gutenberg University, Mainz, Germany

• Christian Hundt
  • chundt@nvidia.com
  • NVIDIA AI Technology Center

• Kai Xu, Weiguo Liu
  • {xukai16@mail., weiguo.liu@sdu.edu.cn}
  • School of Software, Shandong University, Jinan, China

• https://github.com/sleeepyjack/warpcore (Apache 2.0 License)