High Performance Filters For GPUs




Hunter McCoy, Steven Hofmeyr, Katherine Yelick, **Prashant Pandey**

THE UNIVERSITY OF UTAH

BERKELEY LAB
Bringing Science Solutions to the World

Berkeley
UNIVERSITY OF CALIFORNIA

Applications in an exascale world

- High Performance Data Analytics (HPDA) is the intersection of **High Performance Computing** (HPC) and **Big Data**
- HPDA applications run on massive systems like **supercomputers**
- **GPUs** power these supercomputers

#1: Frontier
9408 nodes, 37,632 GPUs
1,685.65 PFlop/s Peak

Metagenomics

Soil sample

Water sample

Human gut

Metagenomics is the study of microbes that inhabit an environment and their interactions.

THE UNIVERSITY OF UTAH

Metagenomic assembly

- Sequences are generated as fragments called **reads**
- Rebuilding DNA strands from the reads is compute/memory intensive

It's like building the puzzle without the picture on the box and there are multiple different puzzles in the same box!

THE UNIVERSITY OF UTAH

MetaHipMer: an exascale metagenomic assembler

Input reads

***K*-mer analysis**

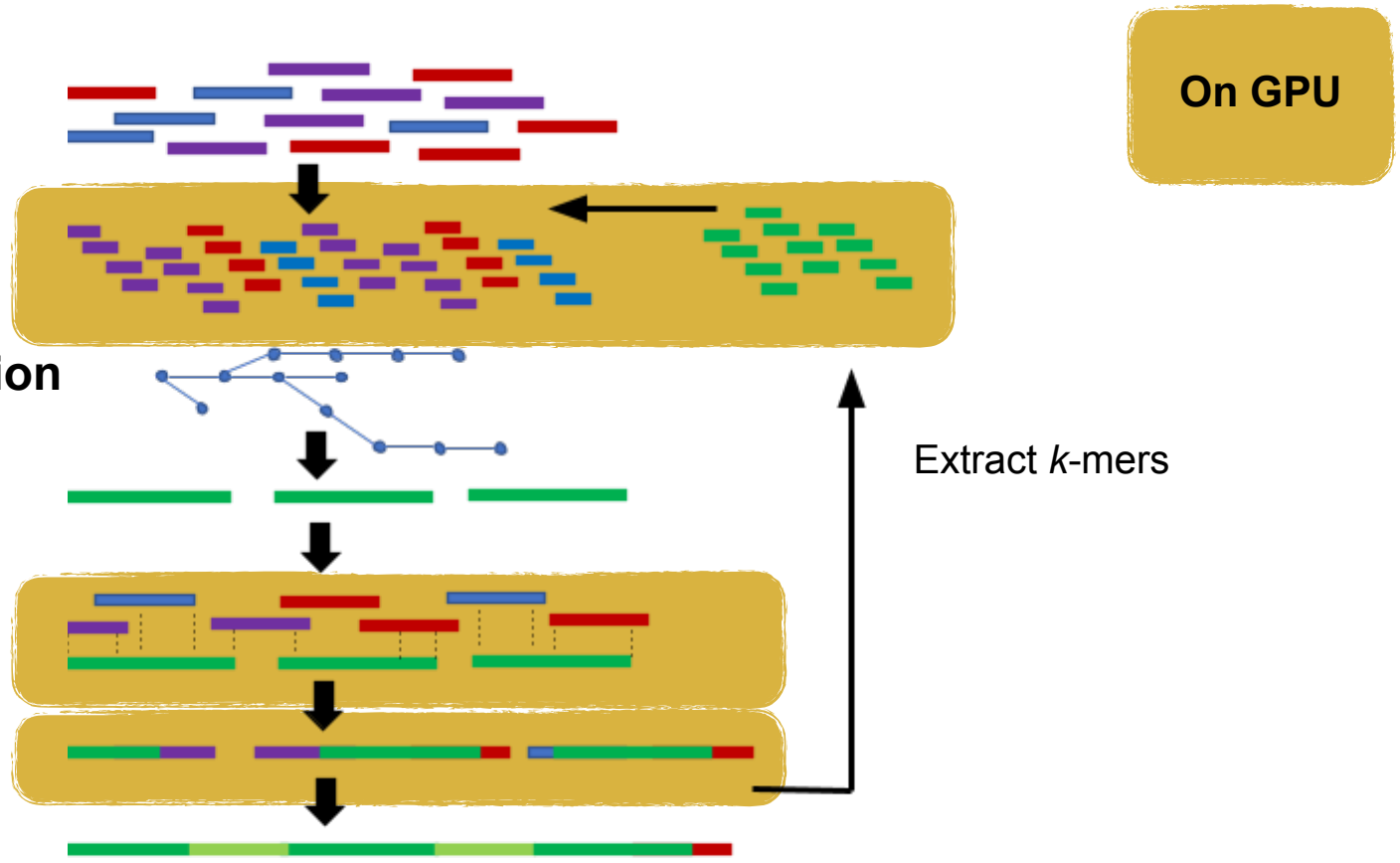
De Bruijn graph construction

Contig generation

Alignment

Local Assembly

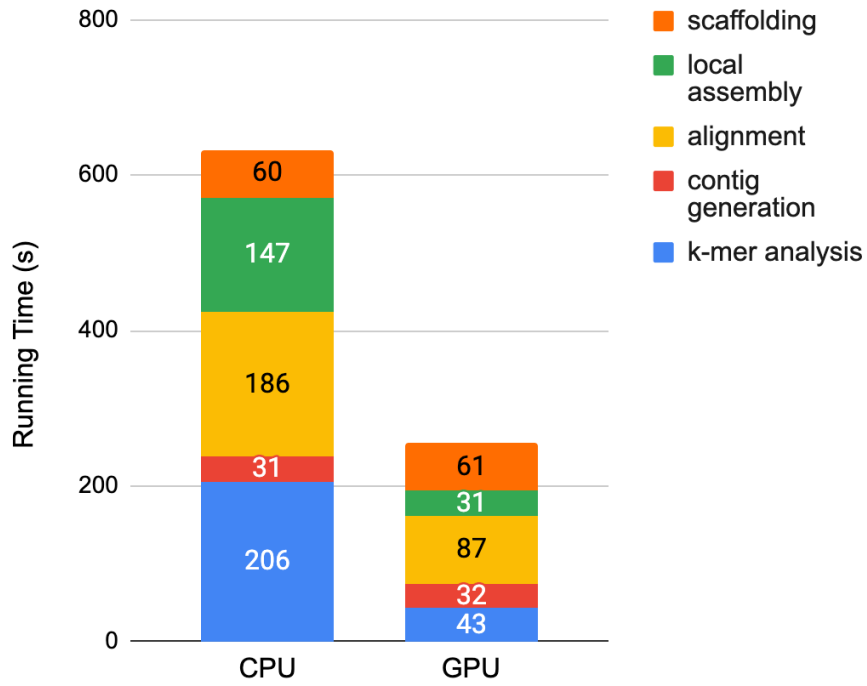
Scaffolding



Hofmeyr, S., Egan, R., Georganas, E. *et al.* Terabase-scale metagenome coassembly with *MetaHipMer*. *Sci Rep* **10**, 10689 (2020).

GPUs accelerate metagenomic assembly

- MHM recently completed the largest co-assembly ever
  - **9,400 nodes** on Frontier
  - **37,000 GPUs**
  - **71.6 terabyte** assembly of Tara Oceans dataset

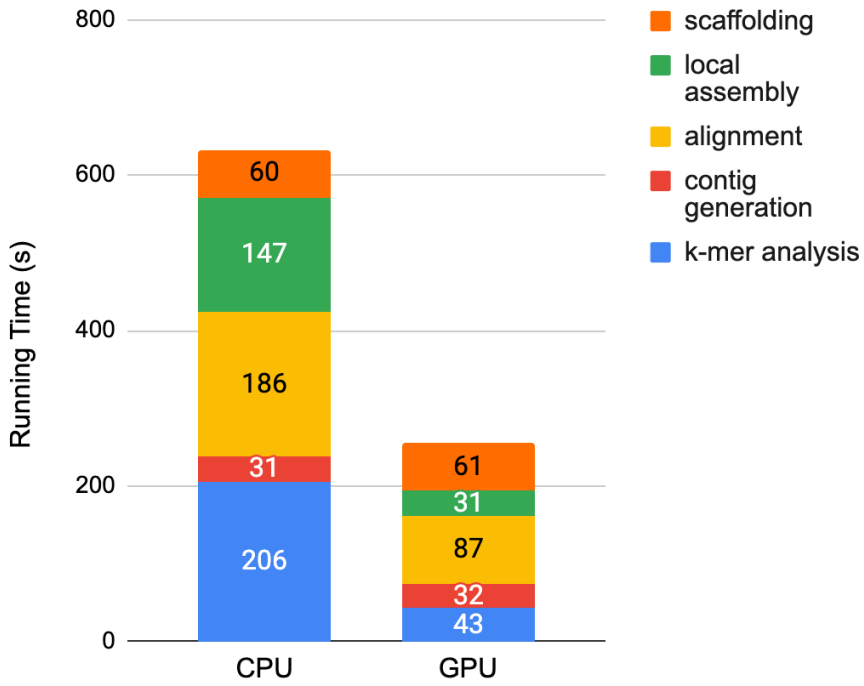


MetaHipMer Running Time

THE UNIVERSITY OF UTAH

GPUs accelerate metagenomic assembly

- MHM recently completed the largest co-assembly ever
  - **9,400 nodes** on Frontier
  - **37,000 GPUs**
  - **71.6 terabyte** assembly of Tara Oceans dataset



MetaHipMer Running Time

105 terabyte Human Microbiome dataset not assembled yet!

THE UNIVERSITY OF UTAH

GPUs are the memory bottleneck!

Input reads

K-mer analysis

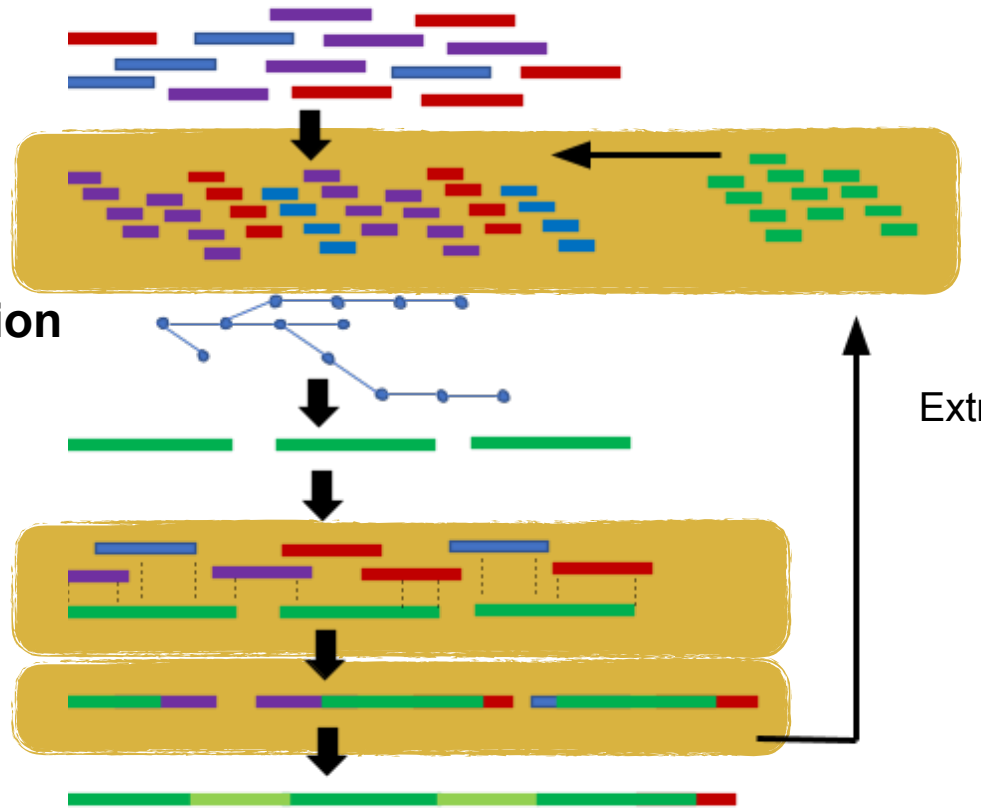
De Bruijn graph construction

Contig generation

Alignment

Local Assembly

Scaffolding



Peak memory usage in *k*-mer analysis!

THE UNIVERSITY OF UTAH

Tradeoff in GPU-enabled *k*-mer analysis

Speed

Memory

Faster compute

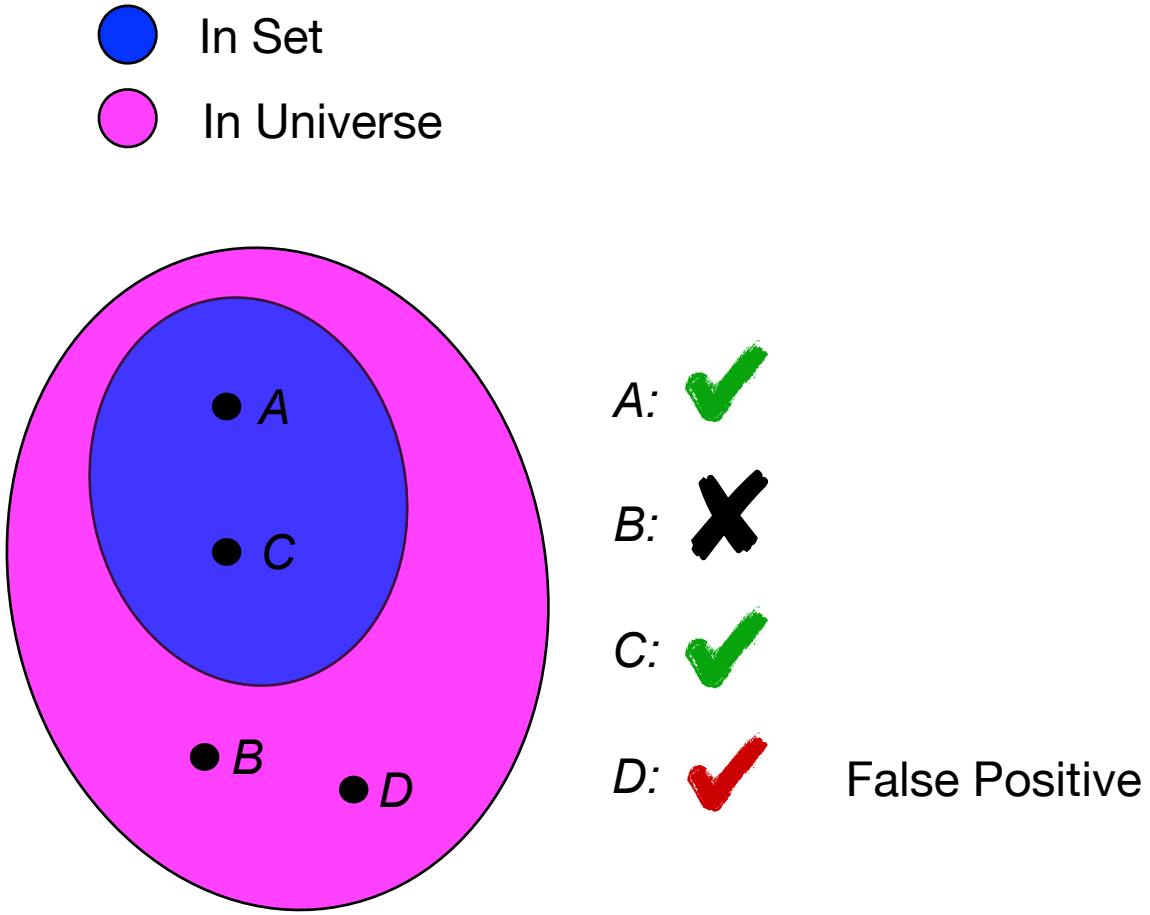
Low device memory

THE UNIVERSITY OF UTAH

Filters can help **overcome** the **memory-speed tradeoff** in **GPUs**!

Filters save space by giving up accuracy

- Filters are a lossy representation of a set, and trade accuracy for space efficiency
- Queries return “maybe” or “definitely not” in set
- False positives occur with bounded error rate ϵ
- Errors are one sided, i.e., no false negatives

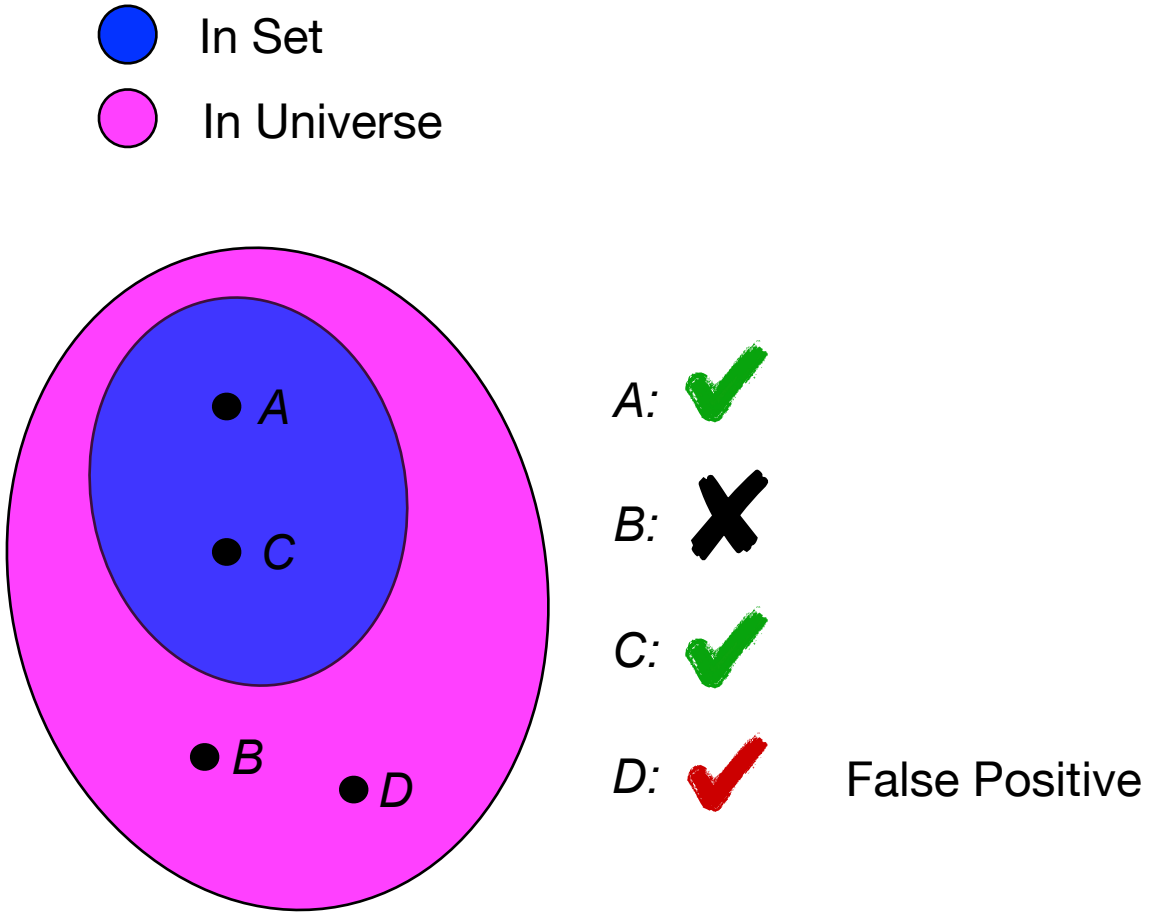


Filters save space by giving up accuracy

- Filters are a lossy representation of a set, and trade accuracy for space efficiency
- Queries return “maybe” or “definitely not” in set
- False positives occur with bounded error rate ϵ
- Errors are one sided, i.e., no false negatives

$$\text{Space} \geq n \log \frac{1}{\epsilon} \text{ bits}$$

For most practical purposes: $\epsilon = 2\,\%$, a filter requires ~8 bits/item

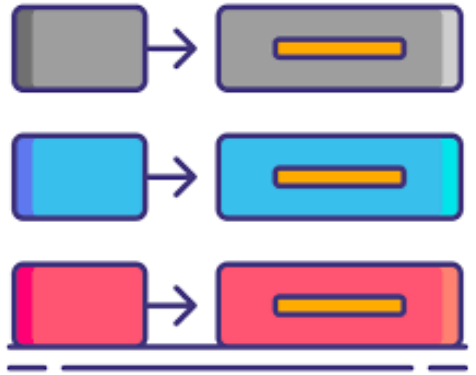


K-mer analysis requires filters with:

High performance

Space efficiency

Deletions



Key-value support

Existing GPU filters lack critical features

	Inserts	Queries	Deletions	Counting	Key-Value Association
Bloom filter	✓	✓			
Blocked Bloom Filter[1]	✓	✓			
RSQF [2]	✓	✓	✓*		✓*
SQF [2]	✓	✓	✓		✓*

THE UNIVERSITY OF UTAH

[1] Junger et al. 2020
[2] Geil et al. 2018

* Not supported in implementation, could be supported in theory

Existing GPU filters lack critical features

	Inserts	Queries	Deletions	Counting	Key-Value Association	Performance
Bloom filter	✓	✓				
Blocked Bloom Filter[1]	✓	✓				✓
RSQF [2]	✓	✓	✓*		✓*	
SQF [2]	✓	✓	✓		✓*	

THE UNIVERSITY OF UTAH

[1] Junger et al. 2020
[2] Geil et al. 2018

* Not supported in implementation, could be supported in theory

Can we build a GPU filter that can achieve
high-performance and **features**?

TCF achieves performance and features

	Inserts	Queries	Deletions	Counting	Key-Value Association	Performance
Bloom Filter	✓	✓				
Blocked Bloom Filter[1]	✓	✓				✓
RSQF [2]	✓	✓	✓*		✓*	
SQF [2]	✓	✓	✓		✓*	
TCF	✓	✓	✓		✓	✓
GQF	✓	✓	✓	✓	✓	✓

THE UNIVERSITY OF UTAH

[1] Junger et al. 2020
[2] Geil et al. 2018

* Not supported in implementation, could be supported in theory

Our results

McCoy, Hofmeyr, Yelick, Pandey PPOPP 2023
McCoy, Hofmeyr, Yelick, Pandey ACDA 2023

- Present new GPU filter designs:
  - Two-Choice Filter (TCF)
    - Stable filter with key-value association/deletion
  - GPU Quotient Filter (GQF)
    - Filter with key-value association/deletion/dynamic counters
- Up to **4.4x faster** than previous GPU filters
- Thread-level point API and host bulk API for easy integration
- **43% reduction in overall peak memory** usage in MetaHipMer

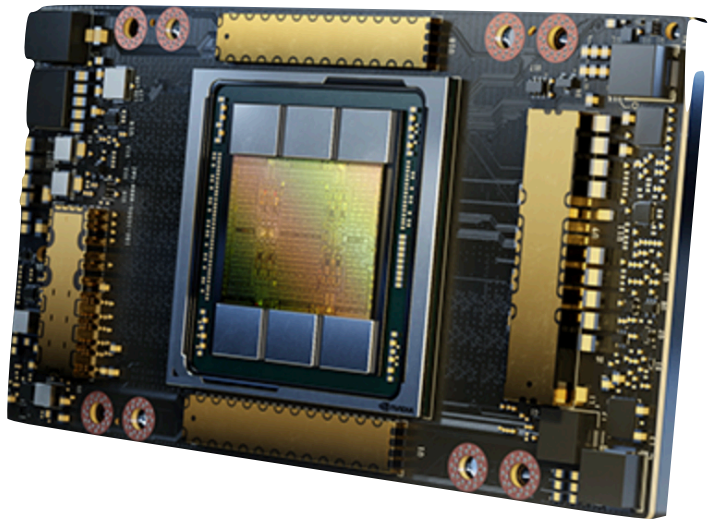
Our results

McCoy, Hofmeyr, Yelick, Pandey PPOPP 2023
McCoy, Hofmeyr, Yelick, Pandey ACDA 2023

- Present new GPU filter designs:
  - **Two-Choice Filter (TCF)**
    - Stable filter with key-value association/deletion
  - GPU Quotient Filter (GQF)
    - Filter with key-value association/deletion/dynamic counters
- Up to **4.4x faster** than previous GPU filters
- Thread-level point API and host bulk API for easy integration
- **43% reduction in overall peak memory** usage in MetaHipMer

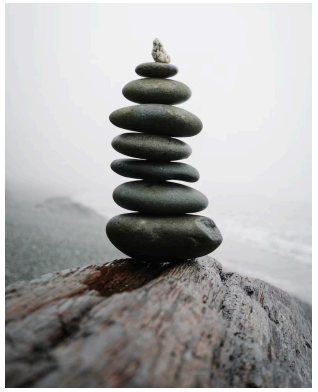
GPU challenges

1. **Thread divergence**
   - Warps diverge and slow down if threads perform different operations
2. **Memory coherence**
   - Warps slow down if threads read from different cache lines
3. **Limited memory**
   - 80 GB vs 1 TB - GPU memory can’t fall back to disk
4. **Massive parallelism**
   - ~80,000-160,000 simultaneous threads



Nvidia A100 Tensor GPU

Design goals for GPU filters



Stability

Items don’t move after insertion

Low associativity

Map each item to one or a small number of locations

Space efficiency

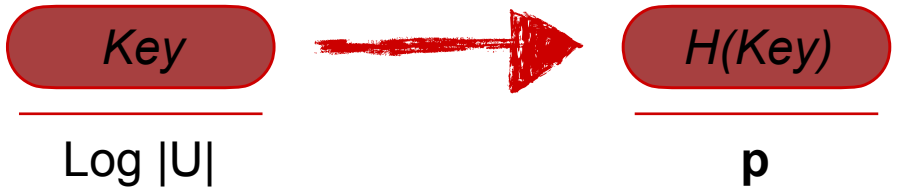
Minimum overhead from pointers or over provisioning

Mapping GPU challenges to filter design goals

Filter design goal		GPU challenge
Low associativity		Thread divergence and memory coherence
Stability		High degree of parallelism
Space efficiency		Limited memory

Fingerprinting is an alternative to Bloom filters

- Filter stores lossy versions of keys called **fingerprints**
  - Fingerprints are **p** bits, stored compactly in a hash table



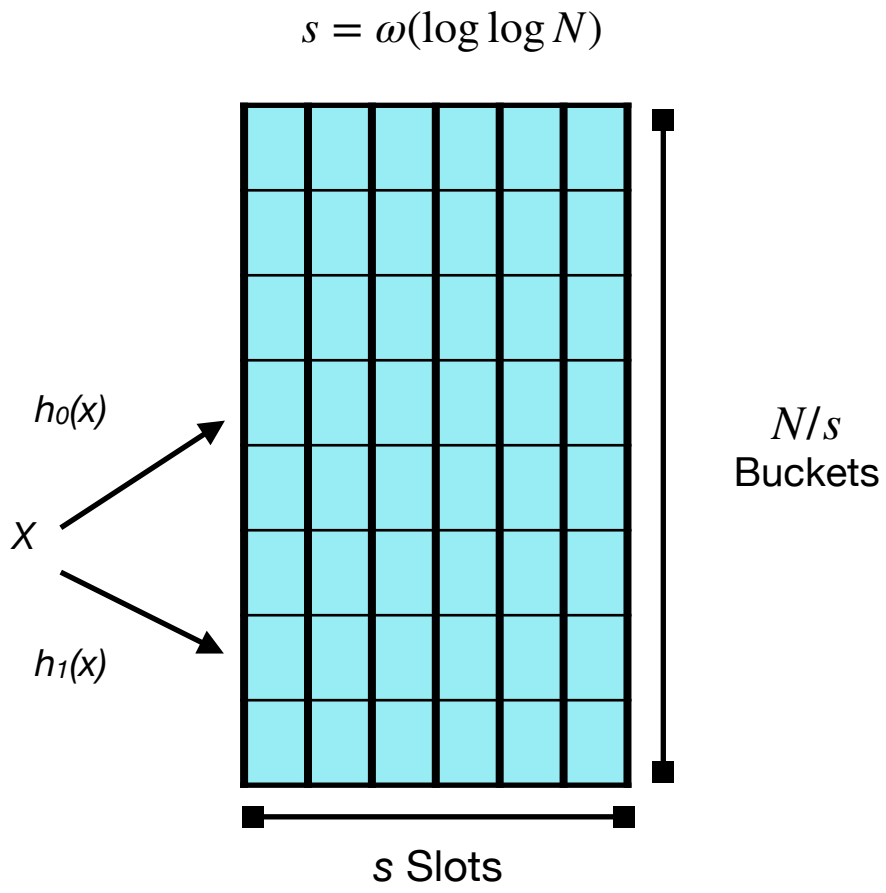
- **Only source of false positives:**
  - Two distinct elements *x* and *y*, where *h(x) = h(y)*

Probability of a collision: $\frac{1}{2^p}$

Two choice filter

Pandey, Conway, Durie, Bender, Farach-Colton, Johnson SIGMOD 2021

- To insert and item *x*
  - Compute $h_0(x)$ and $h_1(x)$
  - Insert $f(x)$ into emptier bucket

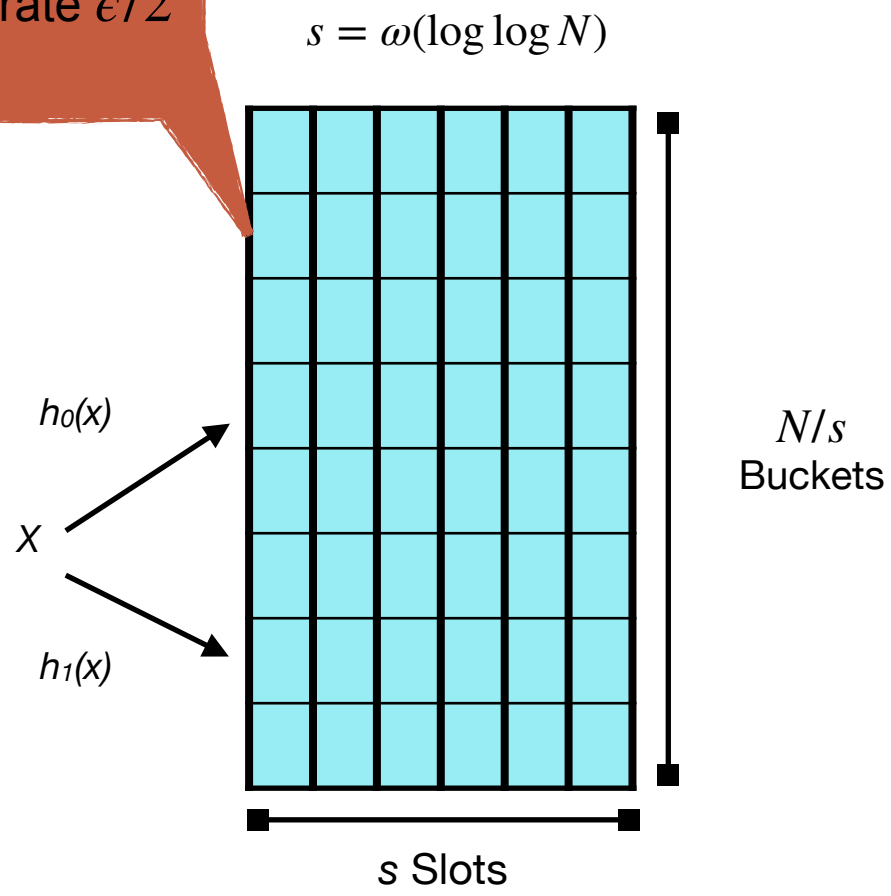


Two choice filter

Pandey, Conway, Durie, Bender, Farach-Colton, Johnson SIGMOD 2021

- To insert and item *x*
  - Compute $h_0(x)$ and $h_1(x)$
  - Insert $f(x)$ into emptier bucket

Each bucket is a mini-quotient filter with false-positive rate $\epsilon/2$ and capacity *s*

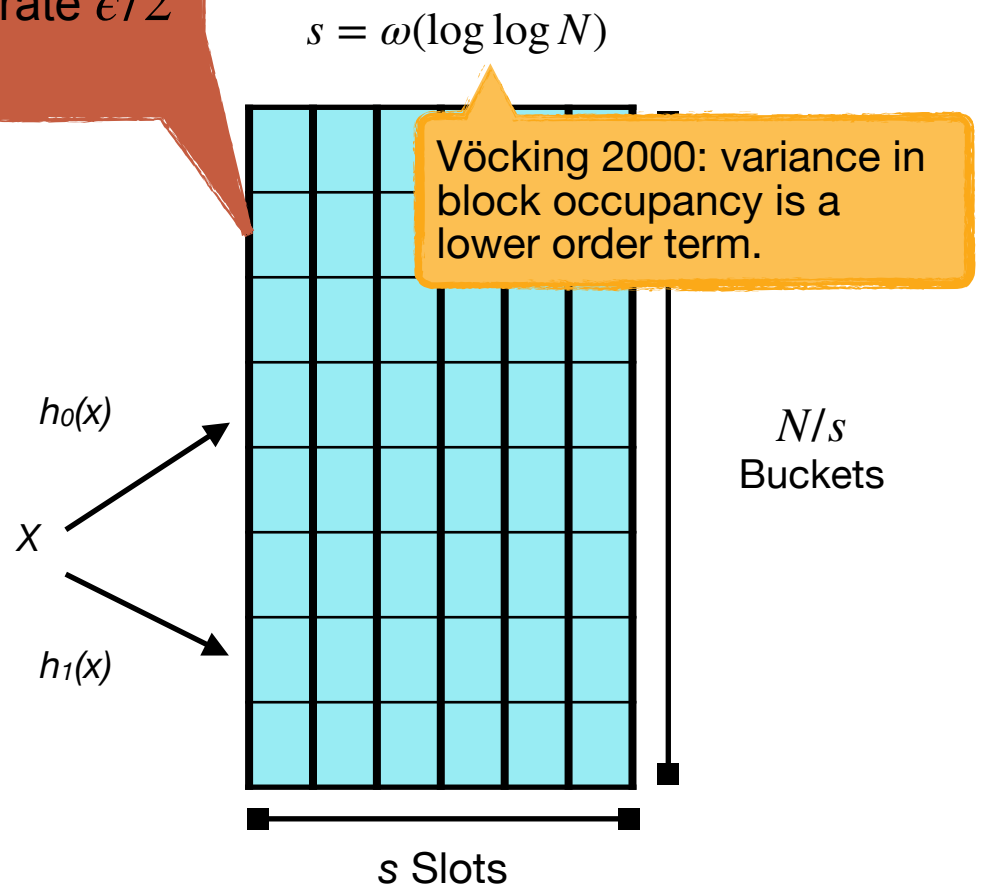


Two choice filter

Pandey, Conway, Durie, Bender, Farach-Colton, Johnson SIGMOD 2021

- To insert and item *x*
  - Compute $h_0(x)$ and $h_1(x)$
  - Insert $f(x)$ into emptier bucket

Each bucket is a mini-quotient filter with false-positive rate $\epsilon/2$ and capacity *s*



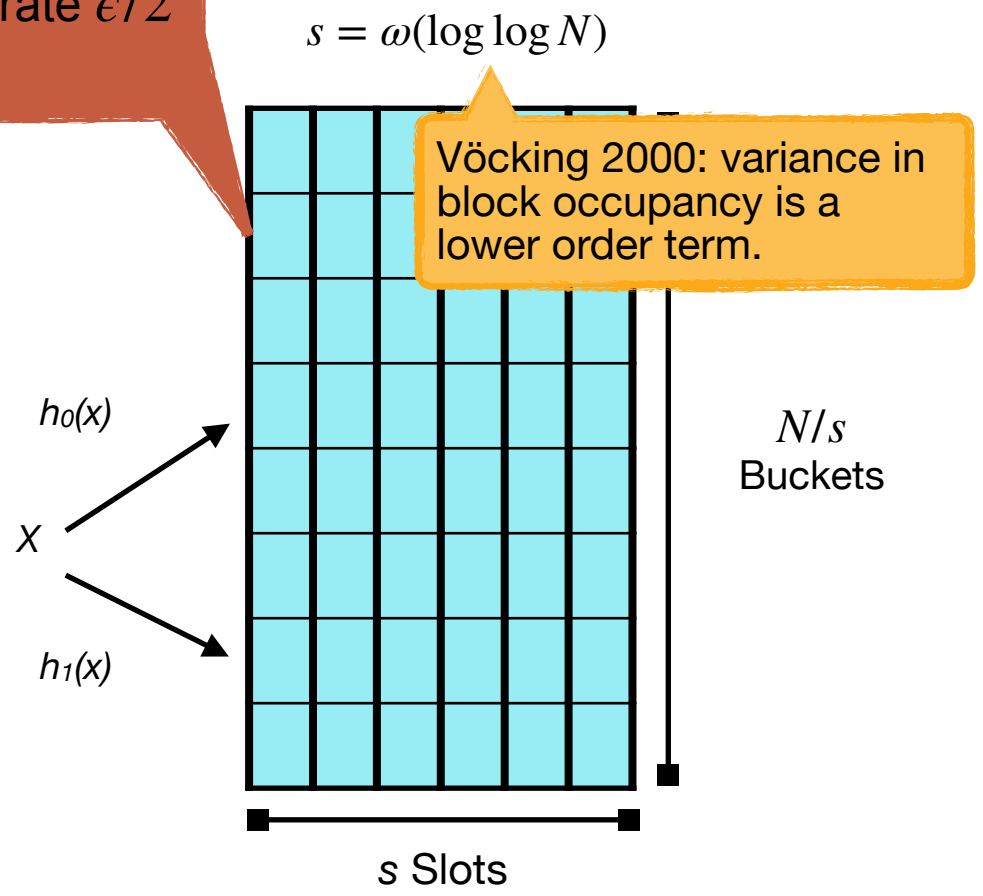
Two choice filter

Pandey, Conway, Durie, Bender, Farach-Colton, Johnson SIGMOD 2021

- To insert and item *x*
  - Compute $h_0(x)$ and $h_1(x)$
  - Insert $f(x)$ into emptier bucket

No kicking across buckets

Each bucket is a mini-quotient filter with false-positive rate $\epsilon/2$ and capacity *s*



Two choice filter

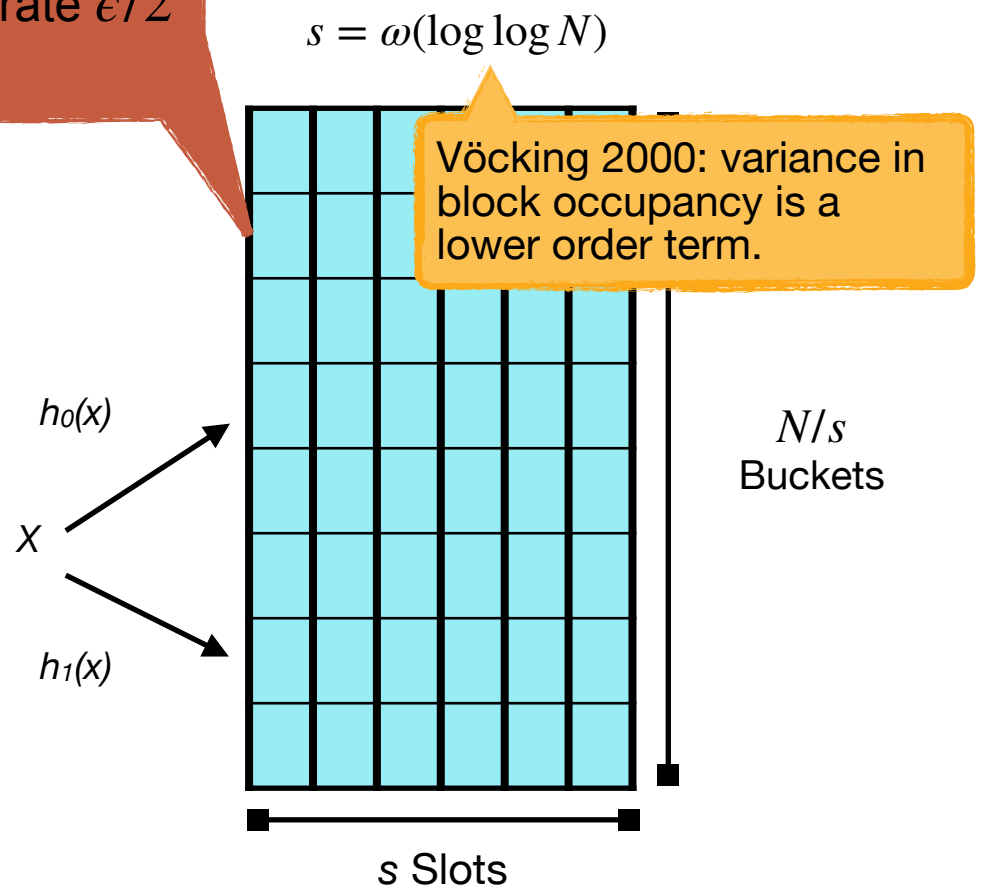
Pandey, Conway, Durie, Bender, Farach-Colton, Johnson SIGMOD 2021

- To insert and item *x*
  - Compute $h_0(x)$ and $h_1(x)$
  - Insert $f(x)$ into emptier bucket

Each bucket is a mini-quotient filter with false-positive rate $\epsilon/2$ and capacity *s*

No kicking across buckets

$h_0(x)$ and $h_1(x)$ can be independent for insert-only



Two choice filter

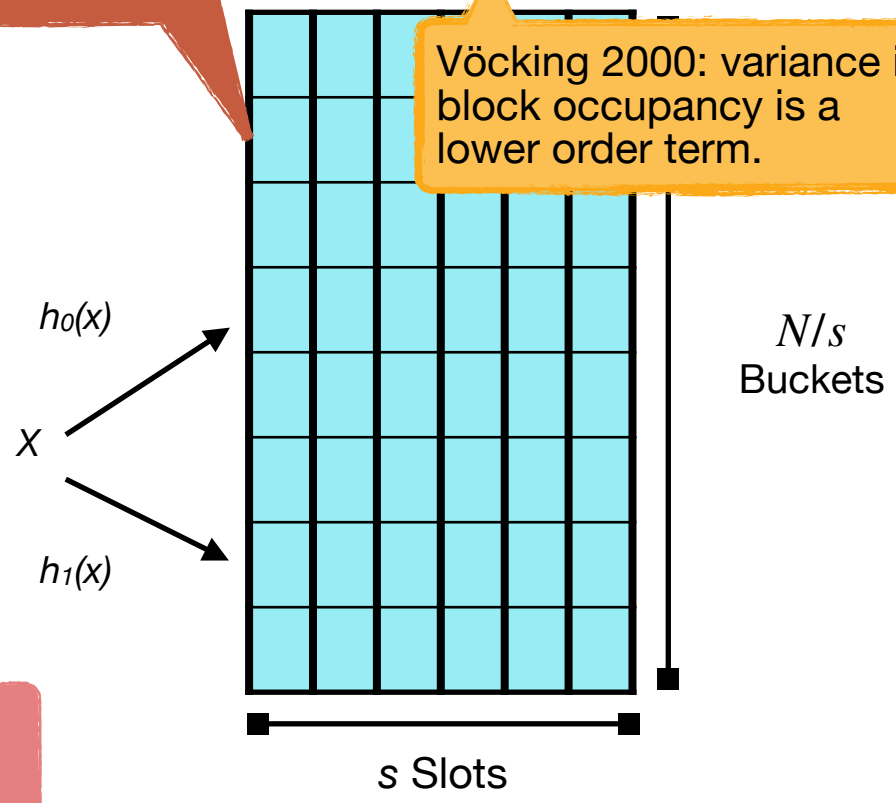
Pandey, Conway, Durie, Bender, Farach-Colton, Johnson SIGMOD 2021

- To insert and item *x*
  - Compute $h_0(x)$ and $h_1(x)$
  - Insert $f(x)$ into emptier bucket

Each bucket is a mini-quotient filter with false-positive rate $\epsilon/2$ and capacity *s*

$s = \omega(\log \log N)$

Vöcking 2000: variance in block occupancy is a lower order term.



No kicking across buckets

$h_0(x)$ and $h_1(x)$ can be independent for insert-only

But still need it to support deletes!

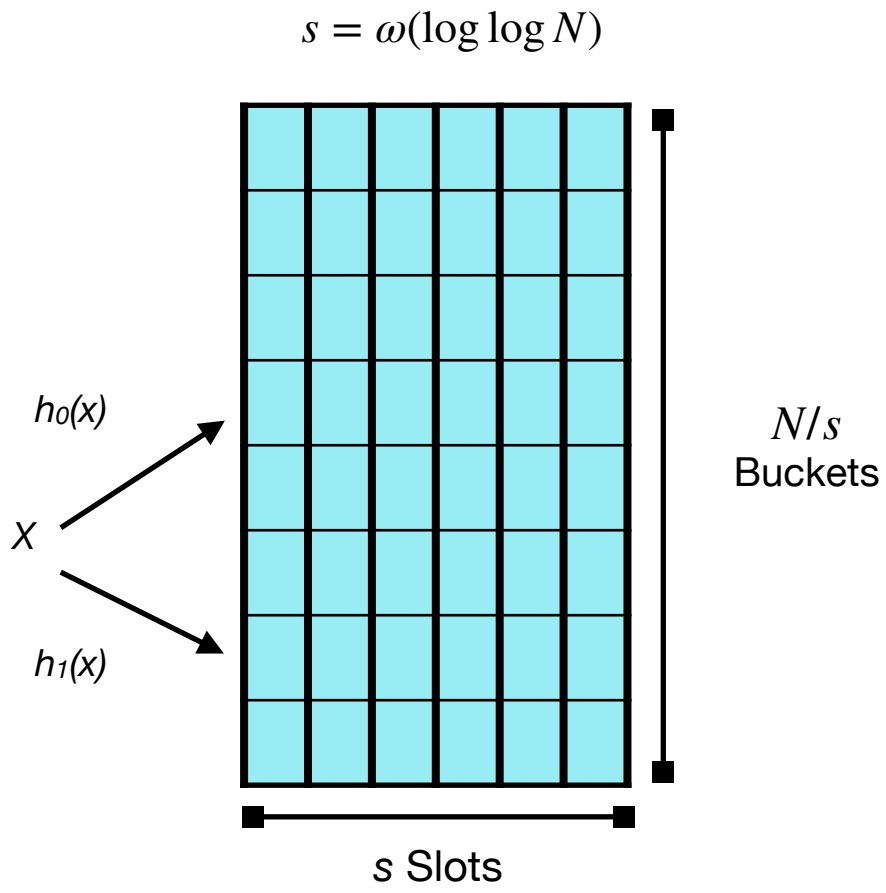
THE UNIVERSITY OF UTAH

Two choice filter on GPUs

Pandey, Conway, Durie, Bender, Farach-Colton, Johnson SIGMOD 2021

- $s = \omega(\log \log N) \approx 48$
  - No drops up to 90% load
  - Strategy used by VQF

- **Slow on GPUs — too many slots to probe with 1 warp**
- **Not stable — tags move inside buckets**
- **Can increase throughput by setting *s* to a smaller value**
  - **However, can't reach high space efficiency**



THE UNIVERSITY OF UTAH

Choosing the optimal bucket size

Can we efficiently use warps with bucket sizes less than 32?

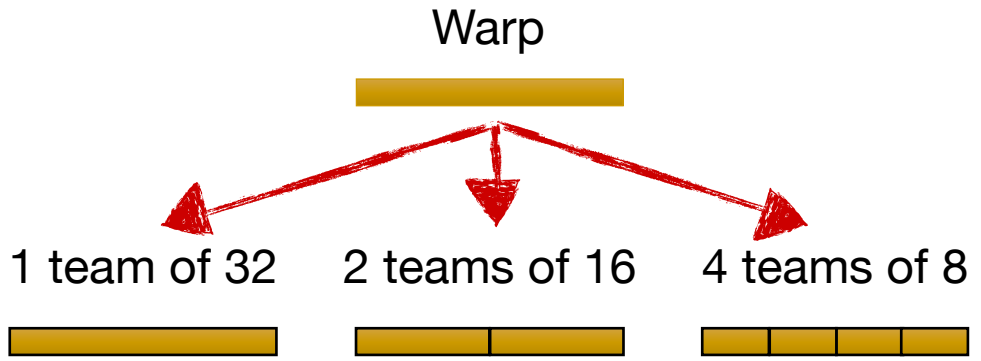
Choosing the optimal bucket size

Can we efficiently use warps with bucket sizes less than 32?

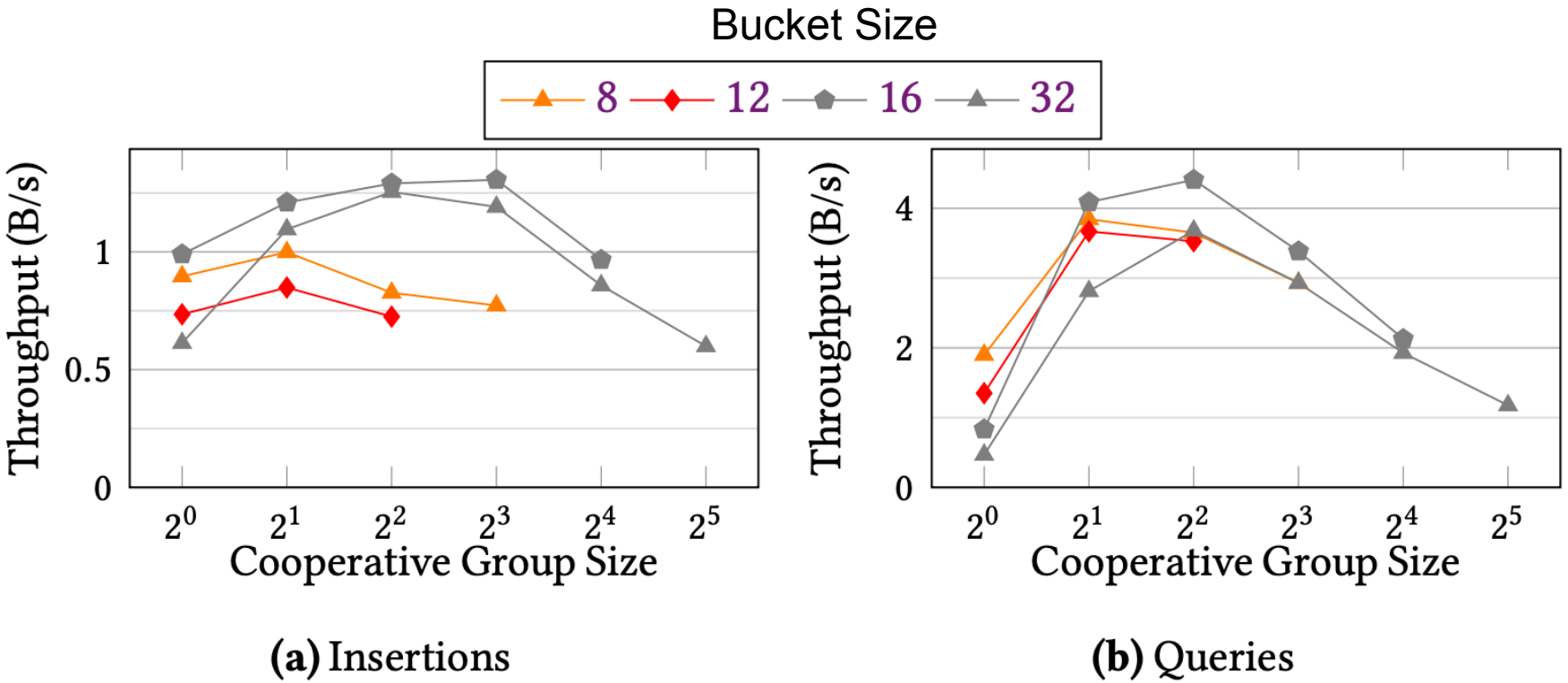
Yes, with Cooperative Groups

With small bucket sizes, warps may not be fully utilized

- The cooperative groups API lets us split warps into smaller teams called **Cooperative Groups**
- This is a logical partition: underlying hardware has not changed
- Cooperative groups let us trade computation for memory:
  - Less compute per group, but we can amortize cost of loading buckets

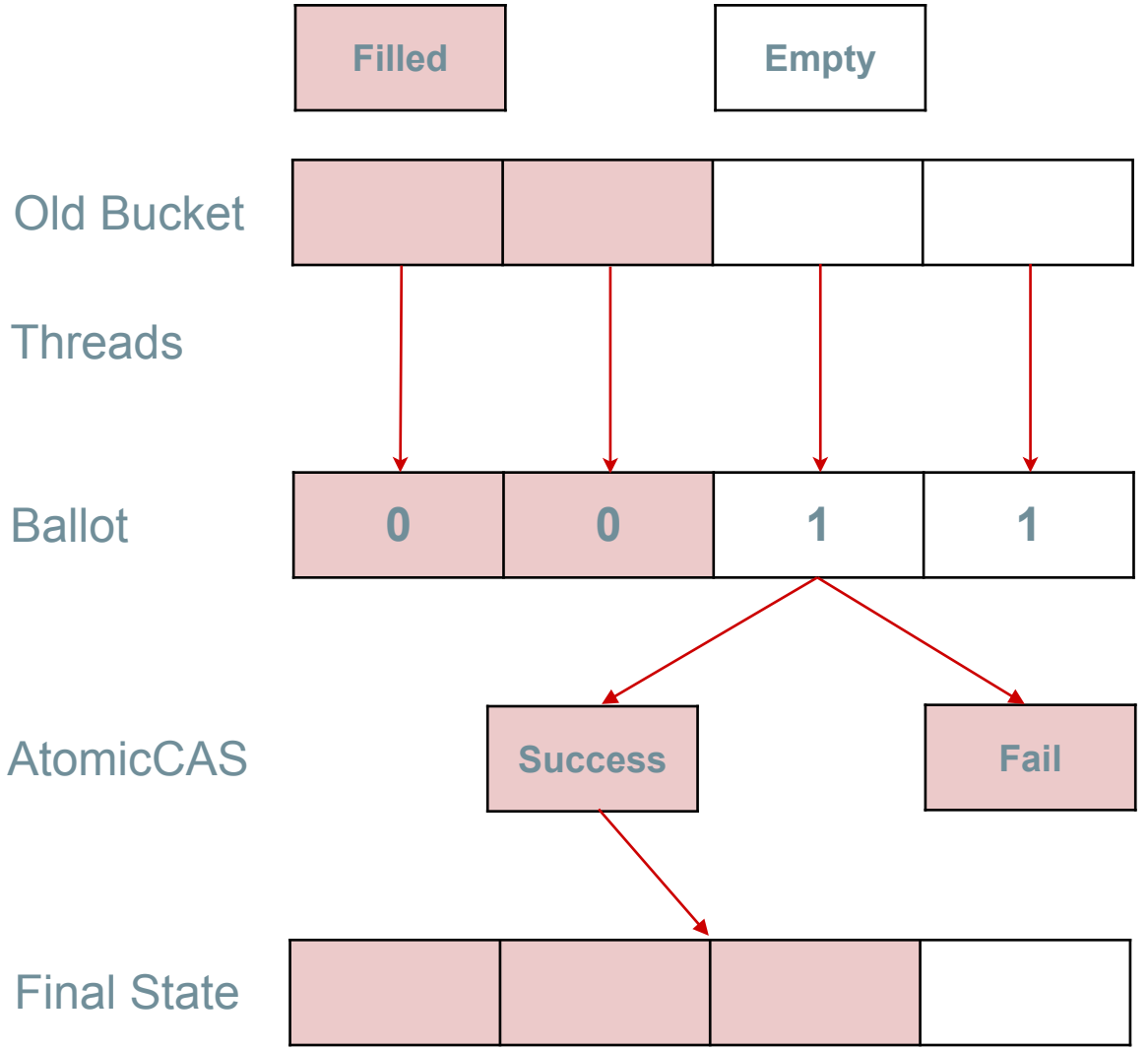


Optimal bucket size



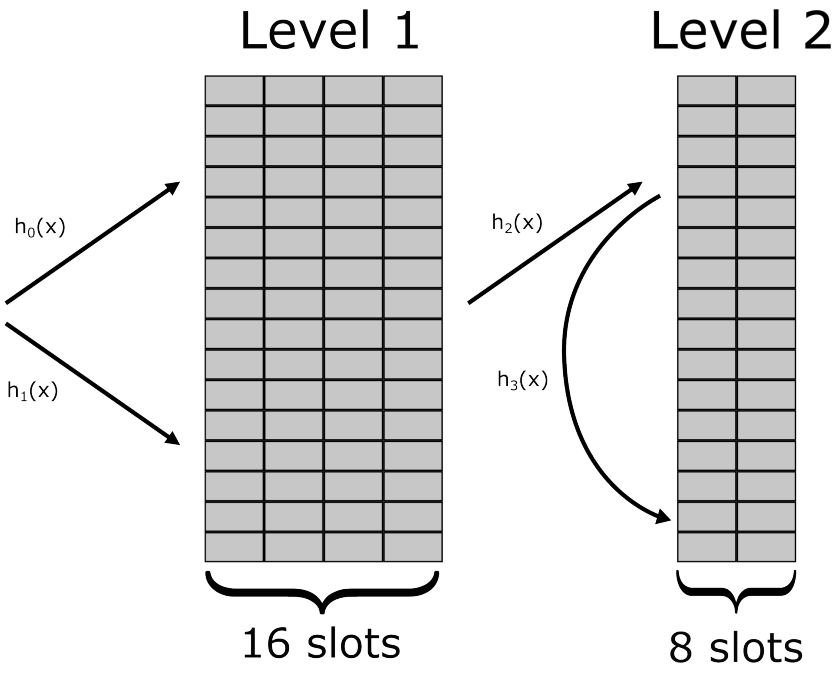
Buckets are modified atomically

- CUDA coherence is weak - no guarantee that changes will be observed in other blocks without thread fencing / atomics
- Cache old state - verify with atomicCAS
- All insertions done atomically, all queries done lazily



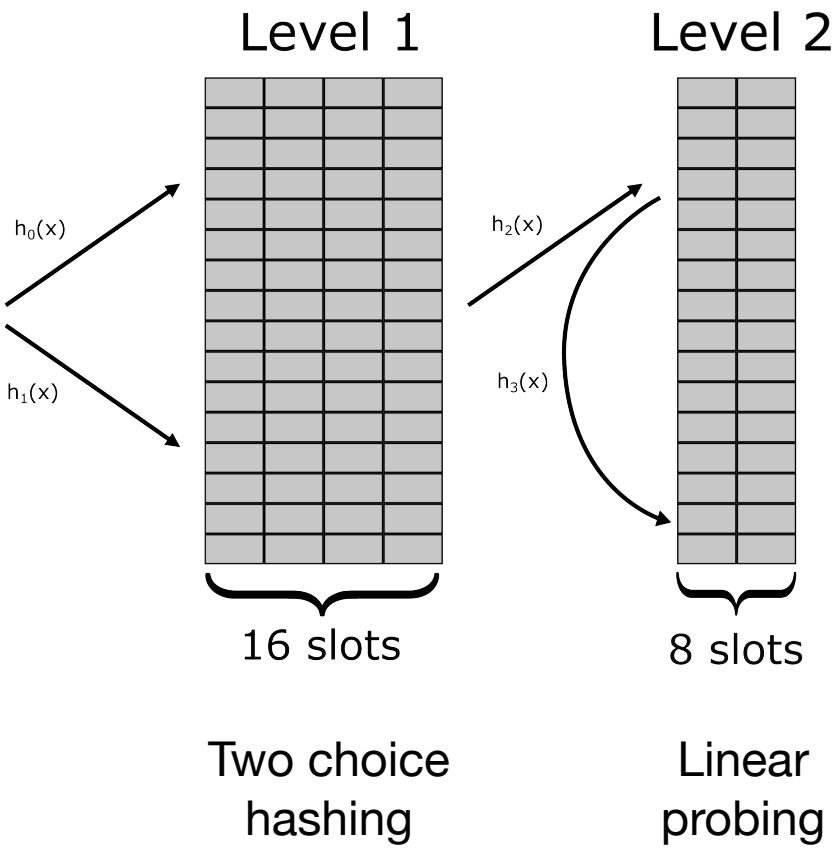
Frontyard-backyard hashing

- Bucket size is chosen to be 16
  - Items drop around 70% load
- Small backing table catches drops, allows scaling to 90% load
  - Backing table is ~1-2% of the total filter size.
  - Uses linear probing to traverse buckets



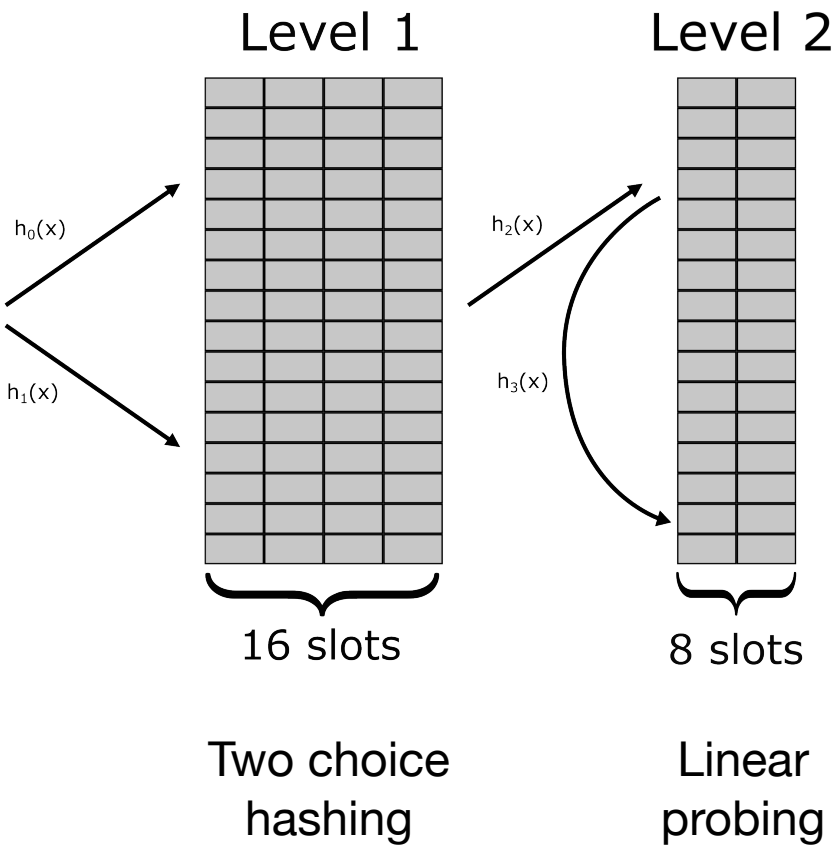
Two-choice filter with backyard

- Modular design with configurable error rate
- Key-value association
- Deletion
- Stable
- Point API for use in kernels.



Two-choice filter with backyard

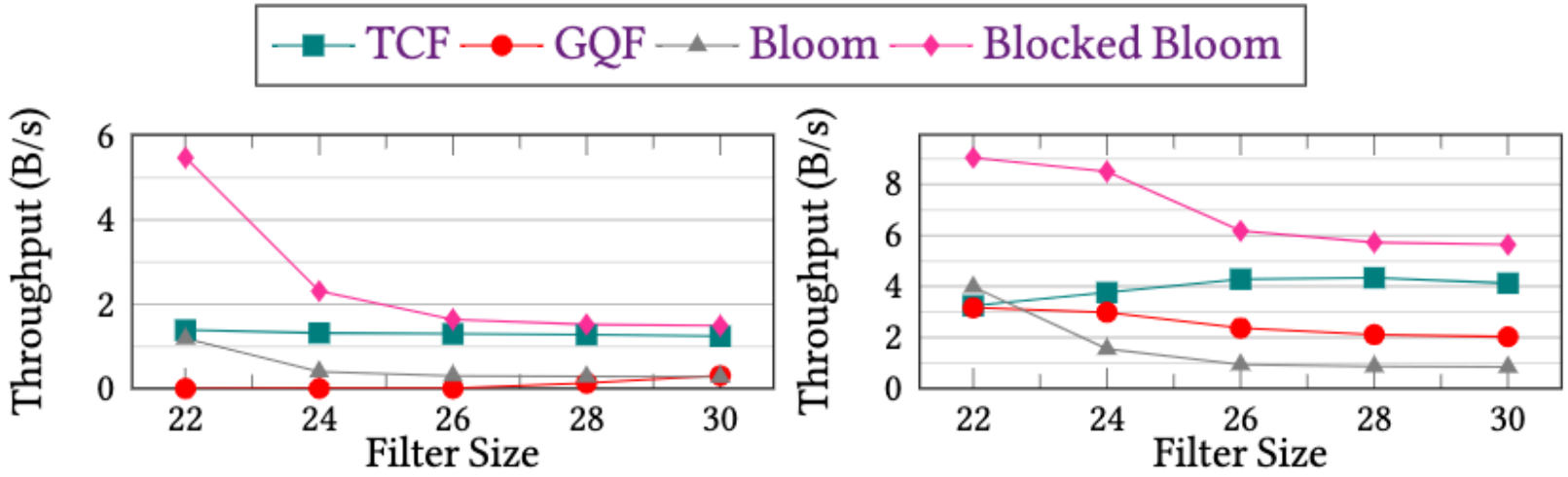
- Modular design with configurable error rate
- Key-value association
- Deletion
- Stable
- Point API for use in kernels.



High performance with features!!

Results

	BF	Blocked BF	SQF	RSQF	TCF	GQF
False Positive (%)	0.15	0.71	1.17	1.55	0.024	0.19
Bits Per Item	10.10	9.73	9.7	7.87	16	10.68

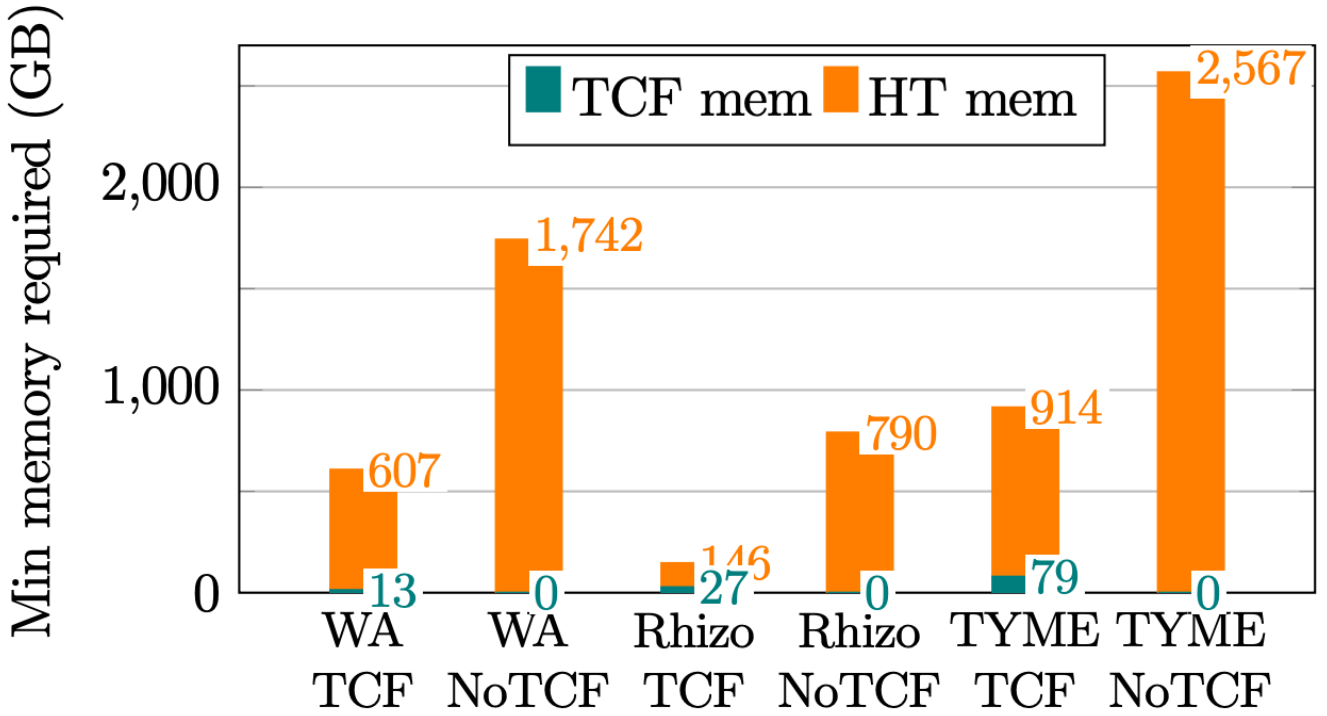


(a) Point Inserts.

(b) Point Queries.

THE UNIVERSITY OF UTAH

Aggregate savings



Peak memory use in *k*-mer analysis is reduced by 2.8 - 5.4x!

This results in a 43% reduction in peak memory use in the assembly pipeline

THE UNIVERSITY OF UTAH

Takeaways

- The two-choice filter overcomes the feature-performance tradeoff of previous GPU Filters
- Simple design with strong theoretical foundation results in practical data structures
- Using a GPU filter can **vastly reduce memory use** of *k*-mer analysis
  - No measured decrease in assembly quality
  - No measured increase in overall runtime
- Filters with advanced features **simplify** the pipeline

Github and lab page:
Libraries: https://github.com/saltsystemslab/gpu-filters
UtahDB: http://mod.cs.utah.edu/

THE UNIVERSITY OF UTAH