GPU Hashing Data Structures and their Application in Accelerated Genomics

Daniel Jünger and Bertil Schmidt

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

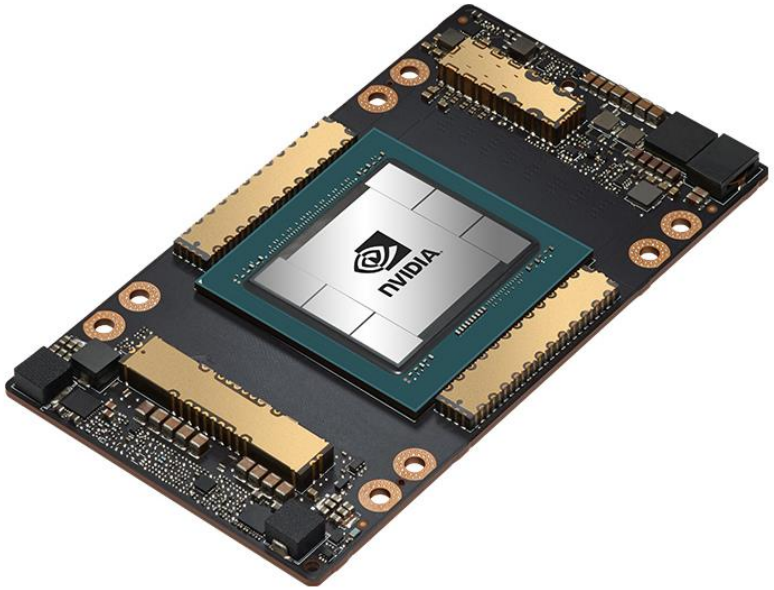
NHR PerfLab Seminar

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:

	Hash Table	Sorted Array	Tree
insertion per element	*O(1)*	*O(log n)*	*O(log n)*
query per element	*O(1)*	*O(log n)*	*O(log n)*
peak memory	*(1 + ε)n*	*2n*	*(1 + ε)n*
final memory	*(1 + ε)n*	*n*	*(1 + ε)n*
range queries	not supported	supported	supported

- 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

Contributions

We propose WarpCore - a versatile library of hashing data structures

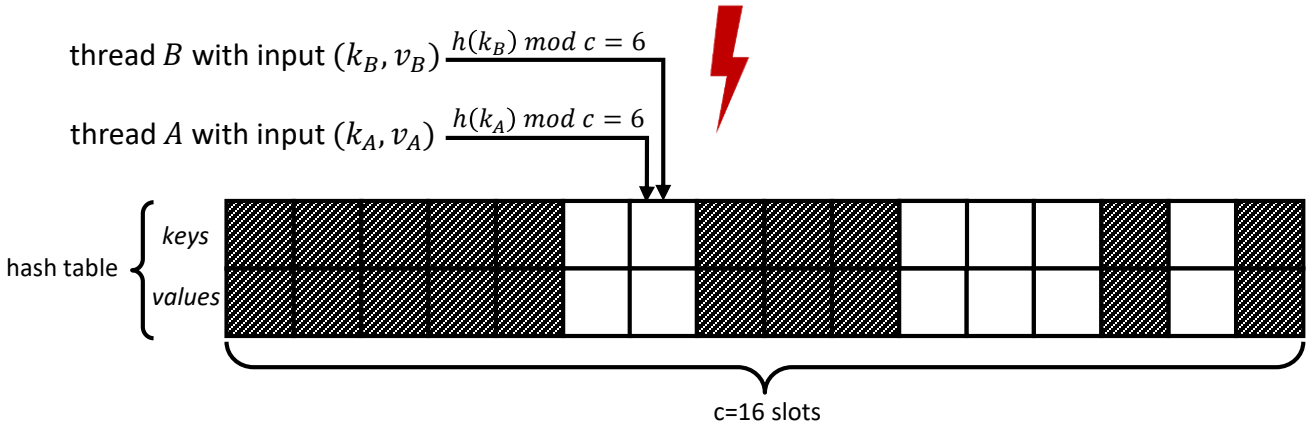
- **Performance**
  - main focus on high-throughput table operations
  - WarpCore outperforms other state-of-the-art CPU and GPU hash tables
- **Modularity**
  - building blocks for constructing customized GPU hash tables
  - probing schemes, hashers, memory layouts, etc.
- **Host-sided and device-sided 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)

Collision Resolution Startegies

Separate Chaining

Slots (buckets) store multiple colliding key-value pairs.

- **Dynamic Linked Lists**
  - allows for dynamic table growth
  - overhead due to memory allocations
  - slow pointer chasing during bucket iteration
- **Static Arrays**
  - memory over-provisioning
  - requires additional array iteration during probing

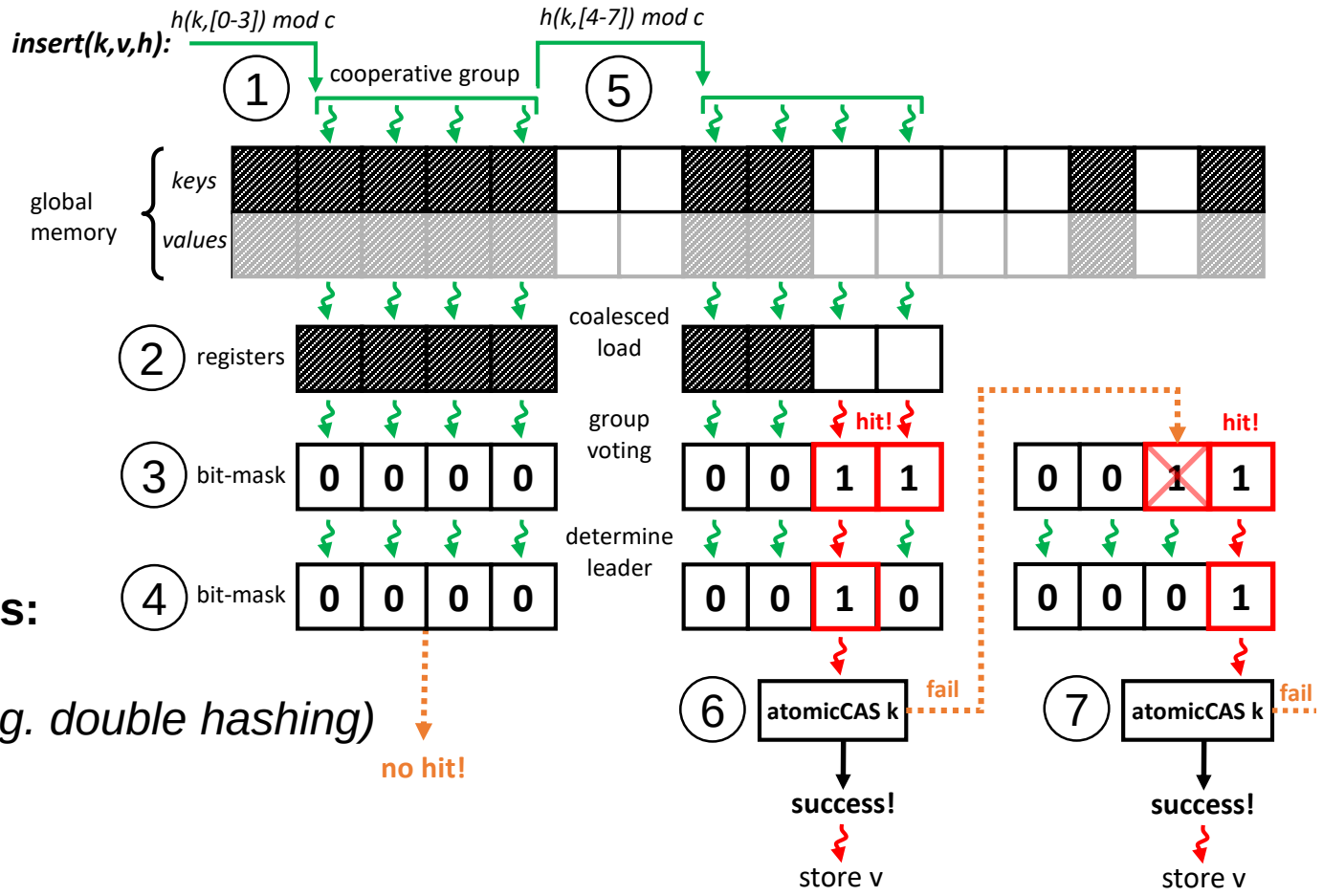
Open Addressing

Find the next unoccupied slot by means of a deterministic probing scheme.

- **Linear Probing:** $\boldsymbol{s(k,i) = (h(k) + i)\ mod\ m}$
  - cache efficient
  - prone to primary and secondary clustering
- **Quadratic Probing:** $\boldsymbol{s(k,i) = \left(h(k) + i^2\right)\ mod\ m}$
  - leaves dense regions faster than linear probing
  - prone to secondary clustering, i.e., $s(k,0) = s(k',0)$
- **Double Hashing:** $\boldsymbol{s(k,i) = \left(h_1(k) + i * h_2(k)\right)\ mod\ m}$
  - if m is prime and $0 < h_2(k) < m$ then
    - $s(k,0) \neq s(k',0)$, i.e., no secondary clustering
    - $s(k,i)$ for $i < m$ is cycle-free
- **Cuckoo Hashing**
  - greedily swap keys between candidate positions
  - may result in infinite cycles
- **Robinhood Hashing**
  - takes from the rich and gives to the poor
  - reduces probing length variance

Cooperative Probing Scheme

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



Considerations for multi-value scenarios:

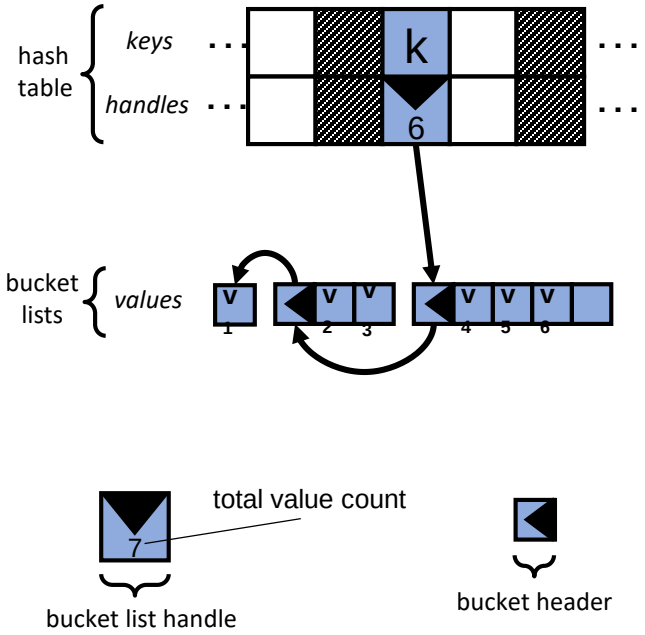
- probing scheme has to be *cycle-free (e.g. double hashing)*
- retrieval can be done cooperatively
- storing identical keys multiple times is memory inefficient

Bucket List Hash Table

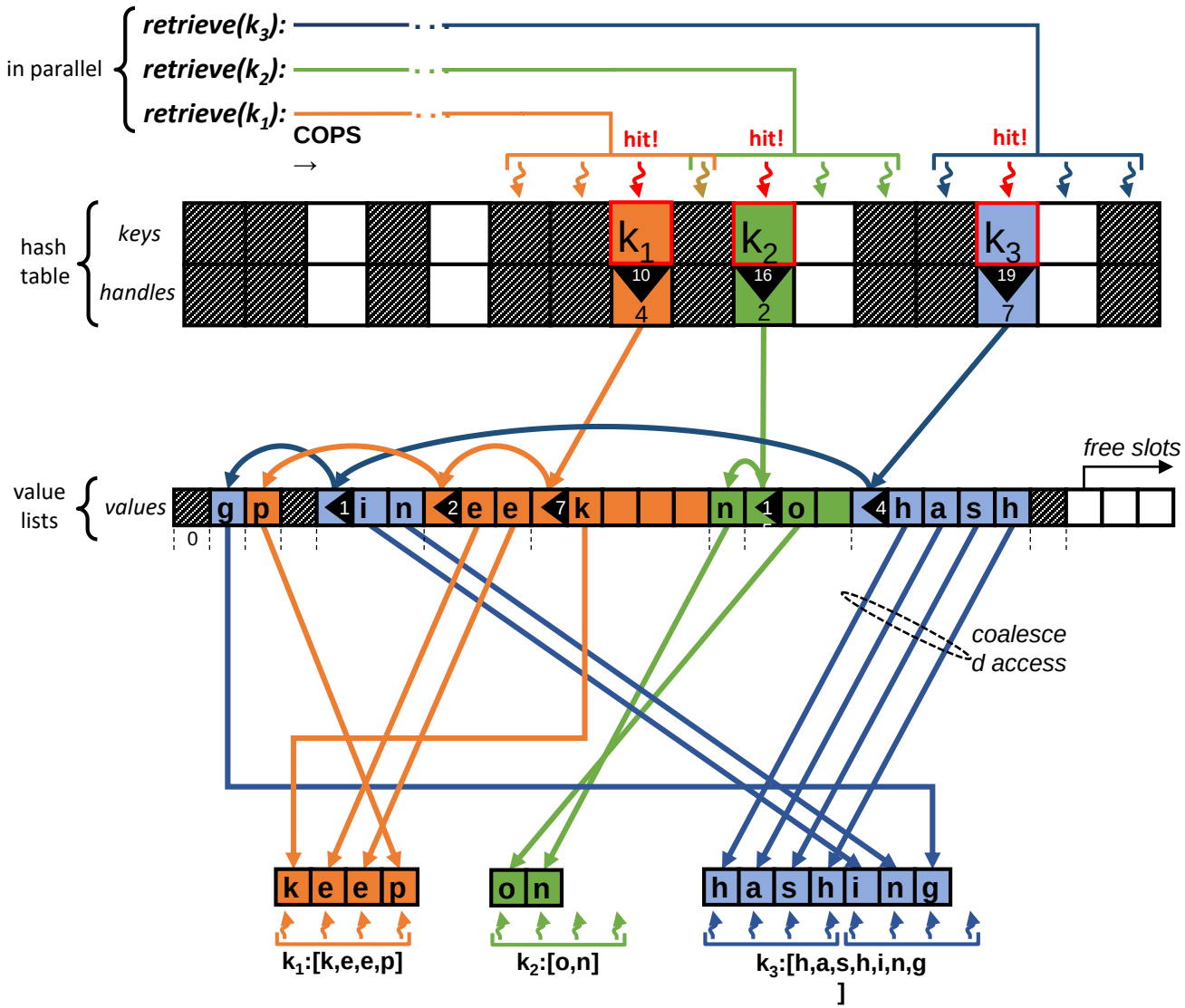
Open adressing hash tables lack space efficiency for highly skewed data

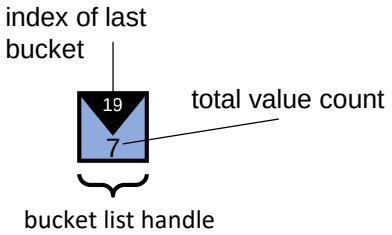
Alternative approach:

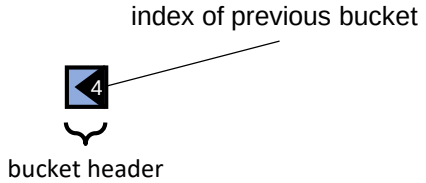
- store keys only once in a single-value OA hash table
- each key holds a handle to a list of values
- each list consists of linked buckets of varying size
- buckets reside inside a pre-allocated memory pool



Bucket List Hash Table



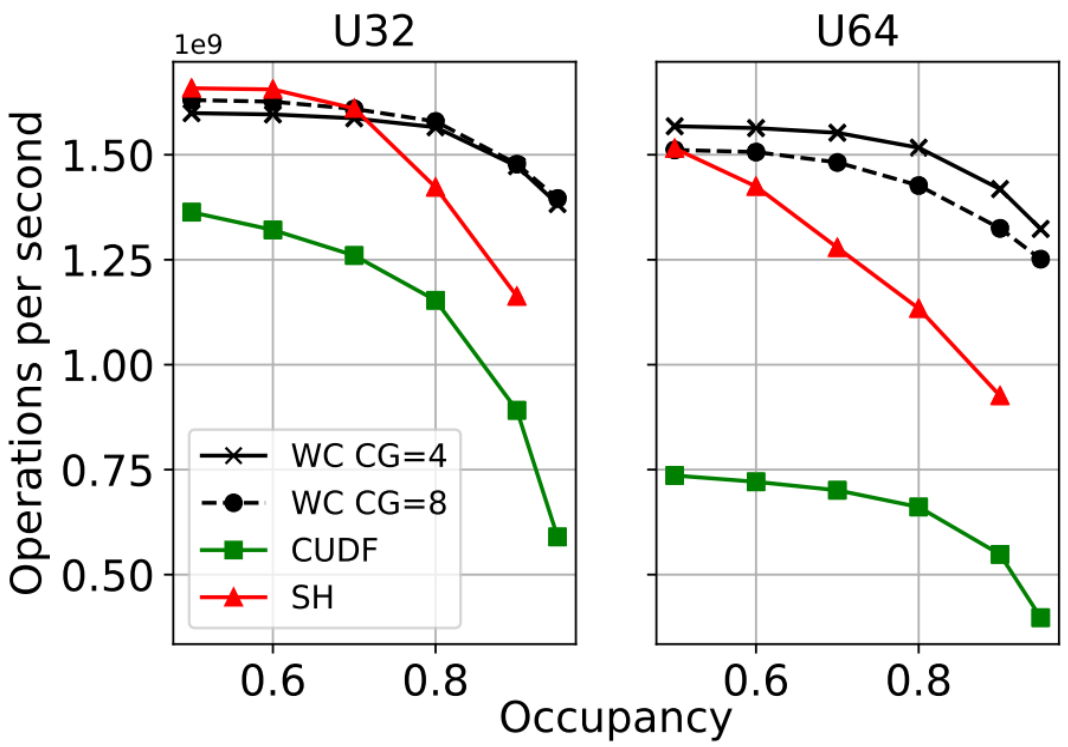




Single-GPU Single-Value Performance

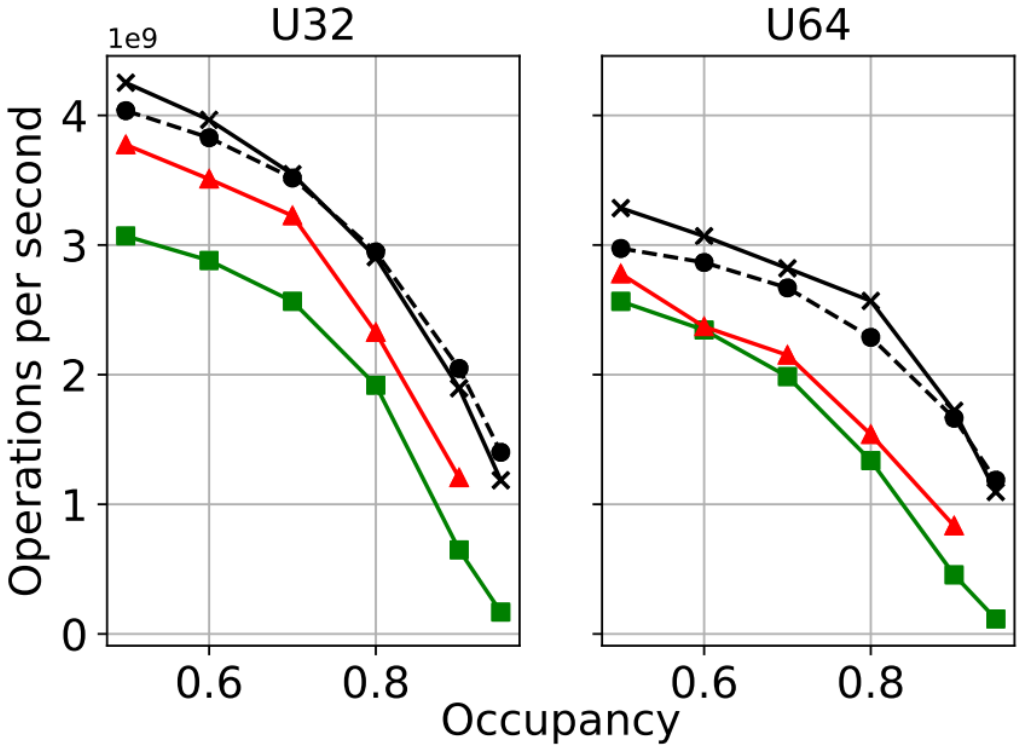
Bulk performance 4+4 byte and (U32) and 8+8 byte (U64) key-value pairs

INSERTION



(a) Tesla V100 (System 1)

RETRIEVAL

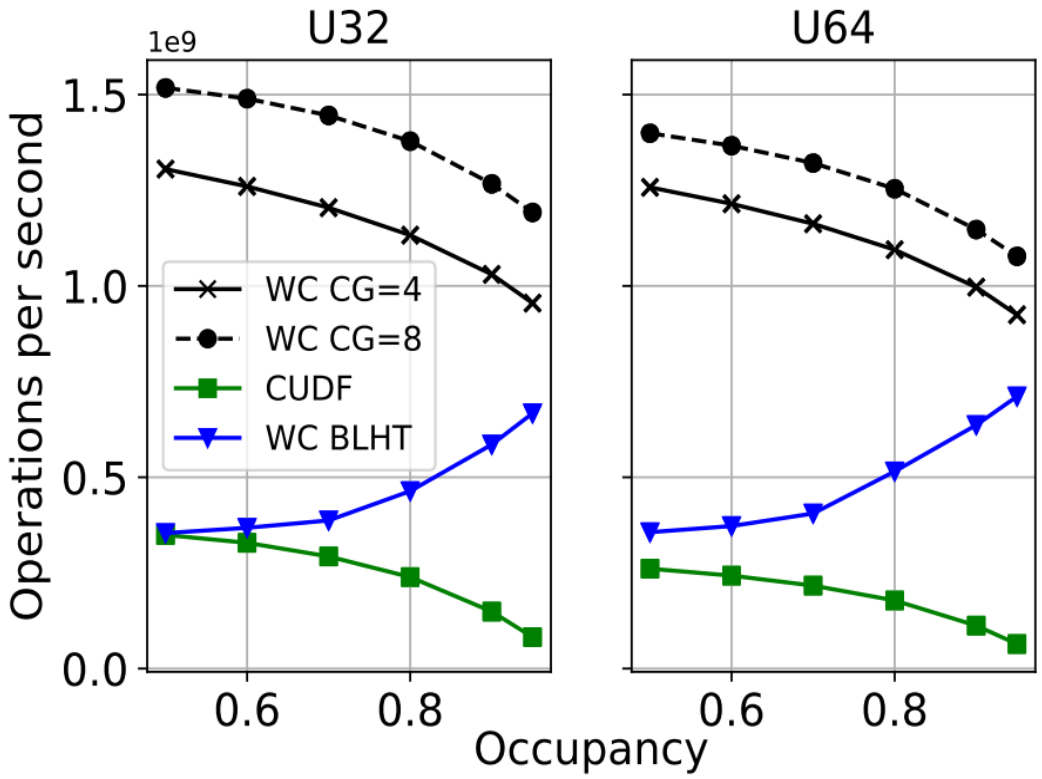


(a) Tesla V100 (System 1)

Single-GPU Multi-Value Performance

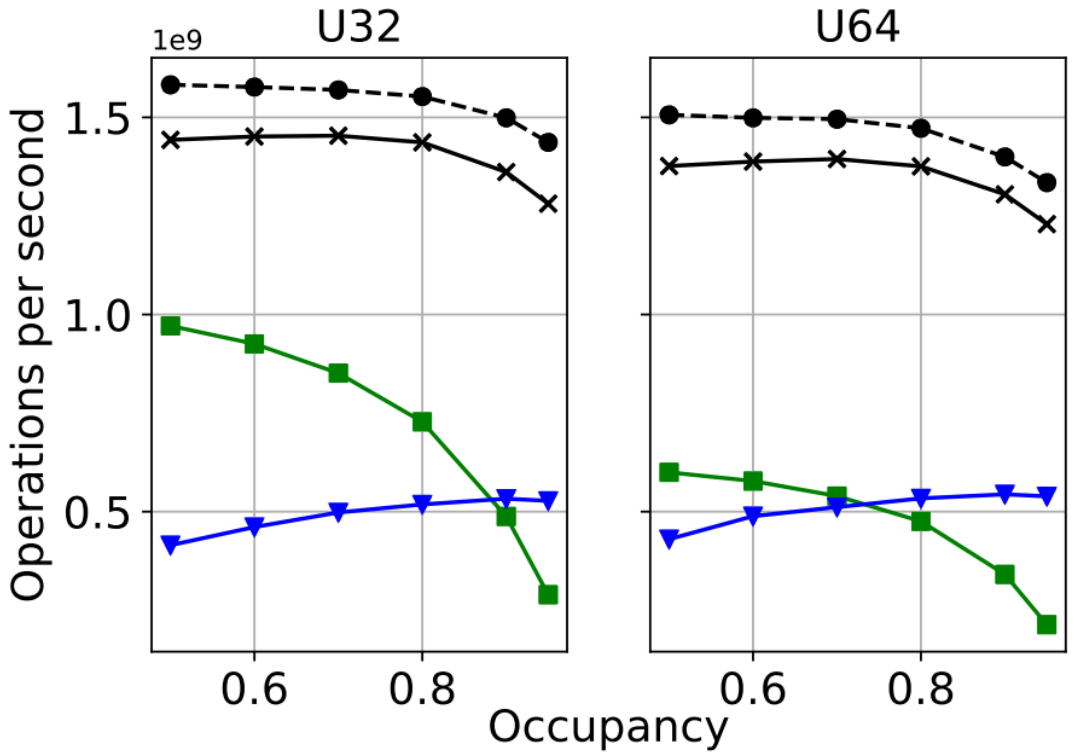
Bulk performance with average key multiplicity of 8

INSERTION



(a) Tesla V100 (System 1)

RETRIEVAL



(a) Tesla V100 (System 1)

Multi-GPU Hashing

random keys

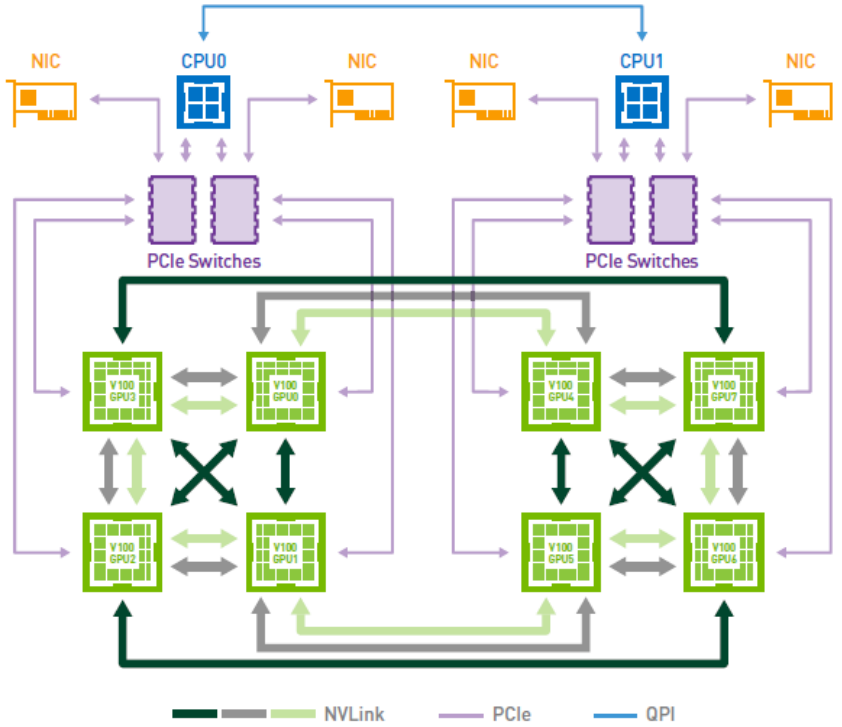
GPU 0	17	40	35	44	56	16	26	∅
GPU 1	12	67	14	94	86	89	53	∅
GPU 2	19	20	37	74	57	73	25	∅
GPU 3	50	11	51	42	31	62	43	∅

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

GPU 0	40	44	56	16	17	26	35	∅
GPU 1	12	89	53	14	94	86	67	∅
GPU 2	20	37	57	73	25	74	19	∅
GPU 3	50	42	62	11	51	31	43	∅

All-to-Allv

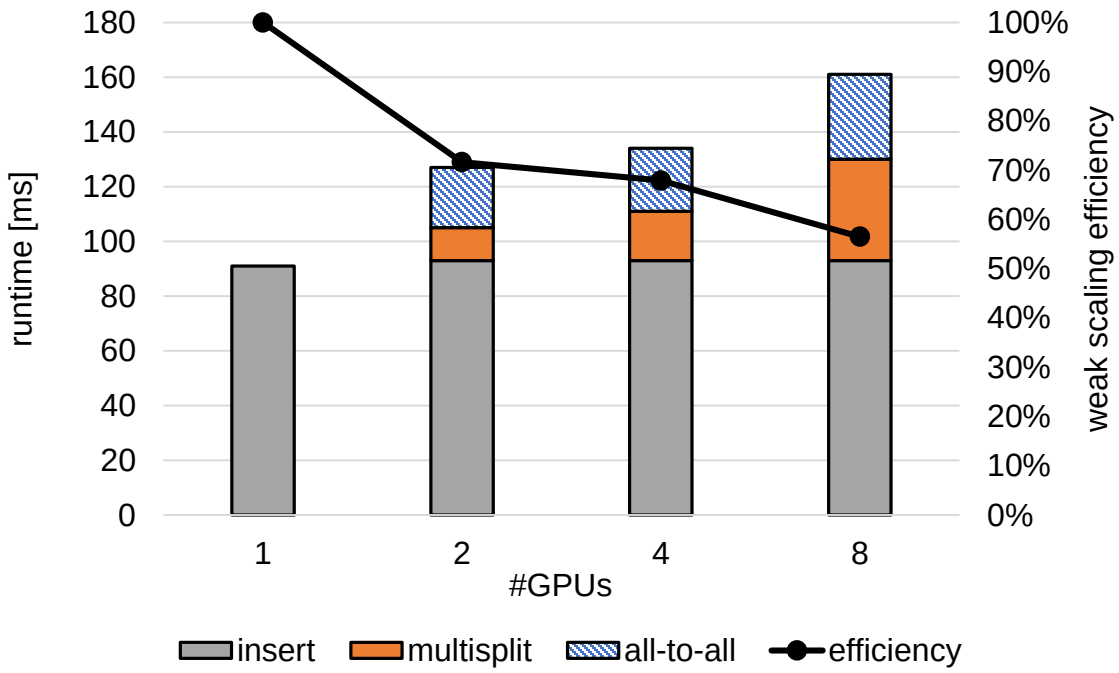
GPU 0	40	44	56	16	12	20	∅	∅
GPU 1	17	89	53	37	57	73	25	∅
GPU 2	26	14	94	86	74	50	42	62
GPU 3	35	67	19	11	51	31	43	∅



- Gossip communication library: 1.8 TB/s (0.5 TB/s) on DGX-2 (DGX-1) for All-to-Allv
  - 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**

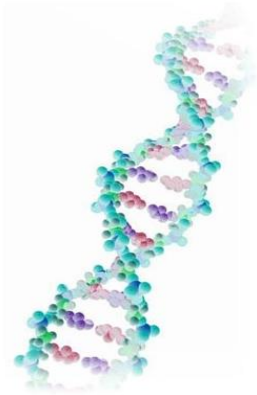
Multi-GPU Single-Value Performance

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)



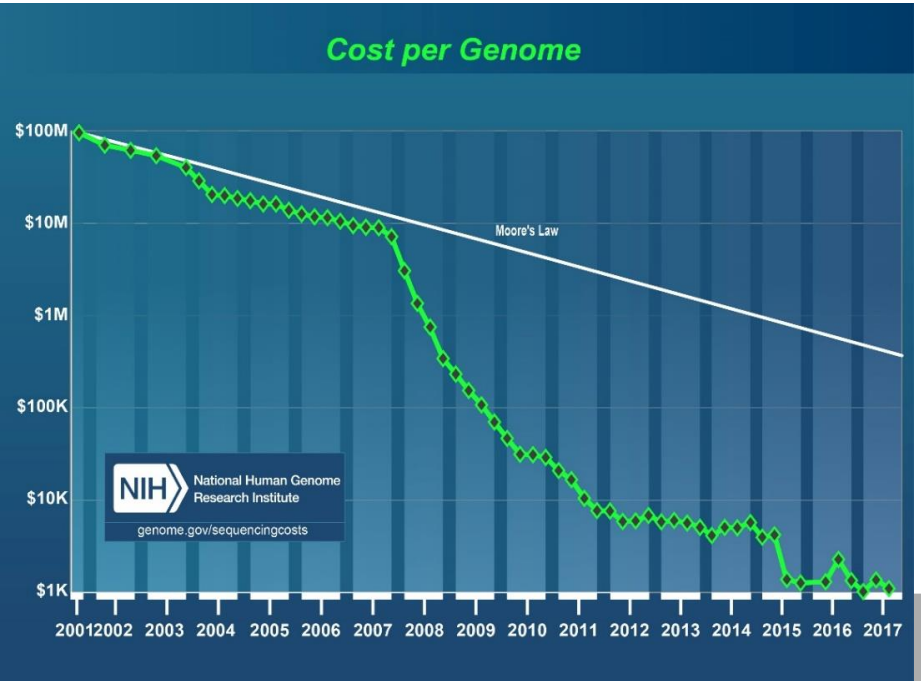
DNA

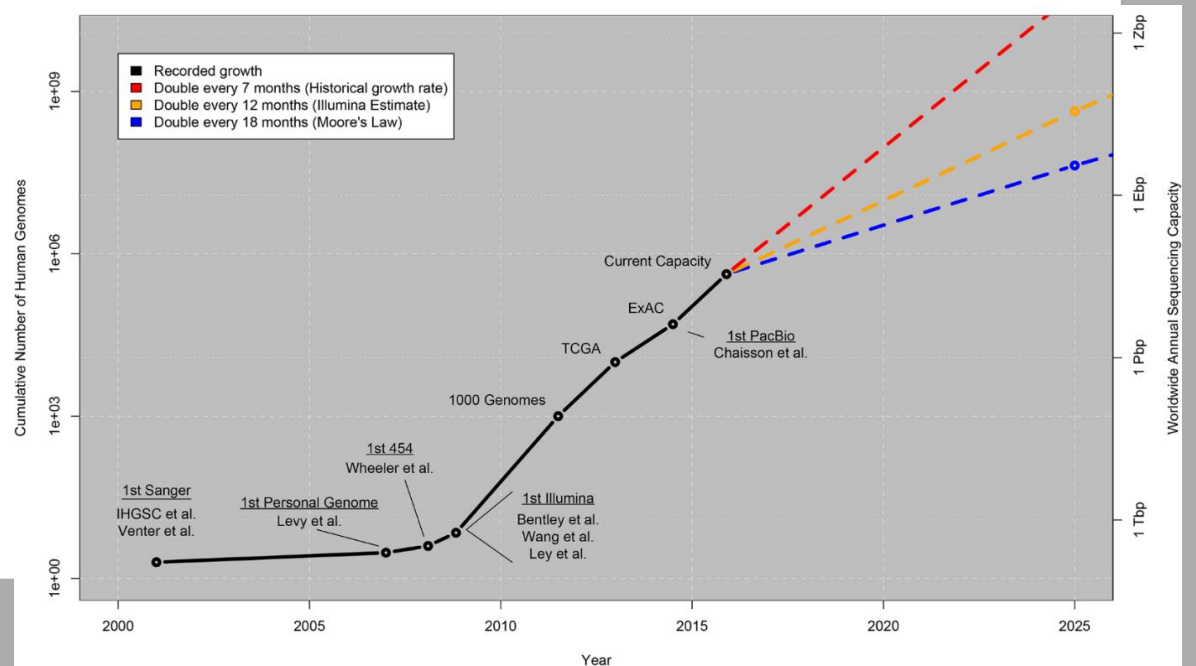
May contain errors!

Read-sequences

Genome seqeunce

Illumina NovaSeq 6000 (Towards $100 per Genome)	
Read length	2×150 bps
Reads per run	20 billion
Run Time	<2 days





Metagenomics

- 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

All-Food-Seq (AFS) Pipeline


Frankfurter Rundschau

Frankfurt | Rhein-Main | Politik | Wirtschaft | Panoram

Übersicht | Landespolitik | Flughafen | Verkehr | Kriminalität | Kirche | OF | HU | DA

Aktuell: Ex-Schülersprecher Bonifer | Bahn-Streik | Europäische Zentralbank

Mainz

Berichte und Bilder von allen wichtigen Ereignissen in Mainz

28. MÄRZ 2013

NAHRUNG

Schneller Blick unter Wurstpelle

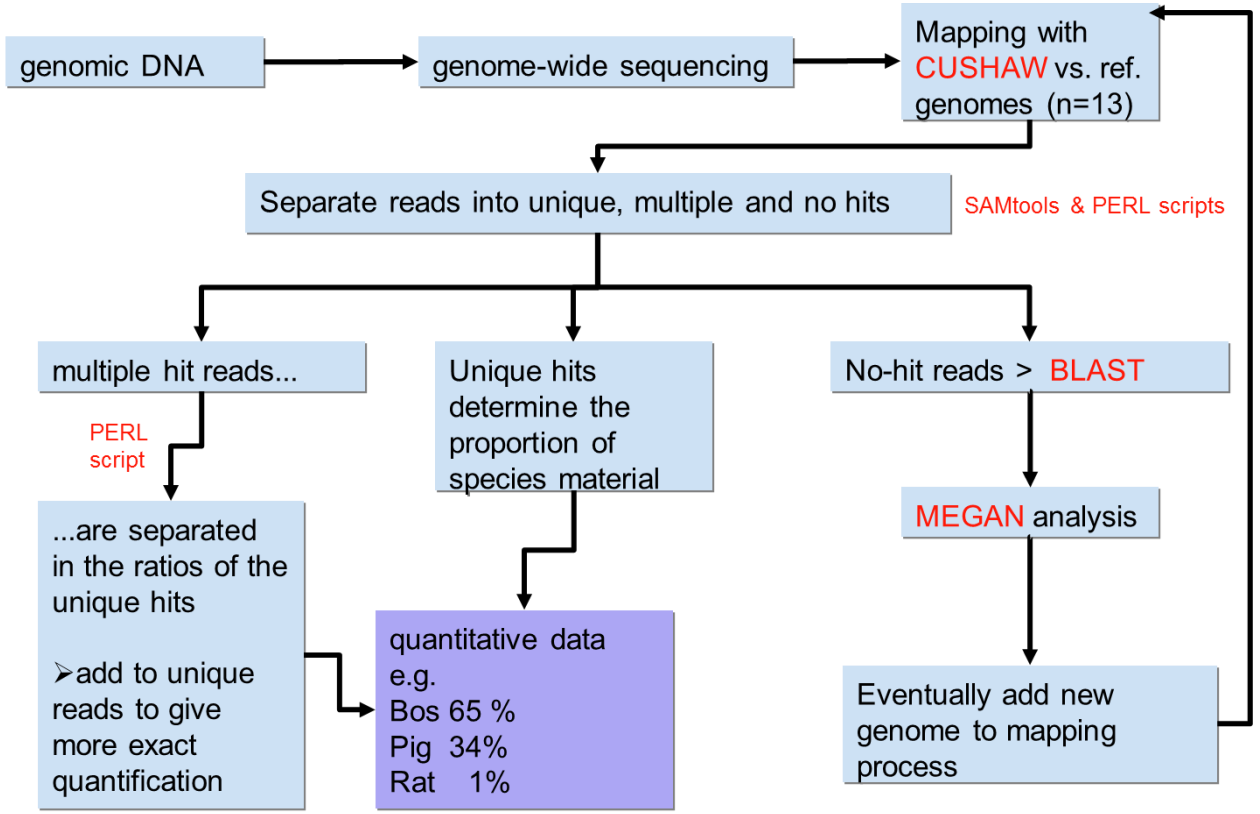
Von ARMIN THOMAS

Was ist drin in der Wurst? Foto: dpa

Wissenschaftler spüren die Zusammensetzung von Nahrung in kleinsten Spuren auf.

Twittern 0 Empfehlen 0 +1 0 per Mail Drucken

MAINZ – Was ist wirklich drin in der Wurst? Mainzer Forscher haben jetzt eine Methode entwickelt, um auch kleinste Spuren von tierischen, pflanzlichen oder mikrobiellen Bestandteilen zu entschlüsseln. Damit können beispielsweise auch Fleischspuren bestimmt werden, die die Lebensmittelüberwachung im Essen gar nicht vermutet hat – angesichts grassierender Lebensmittelskandale ein Fortschritt.

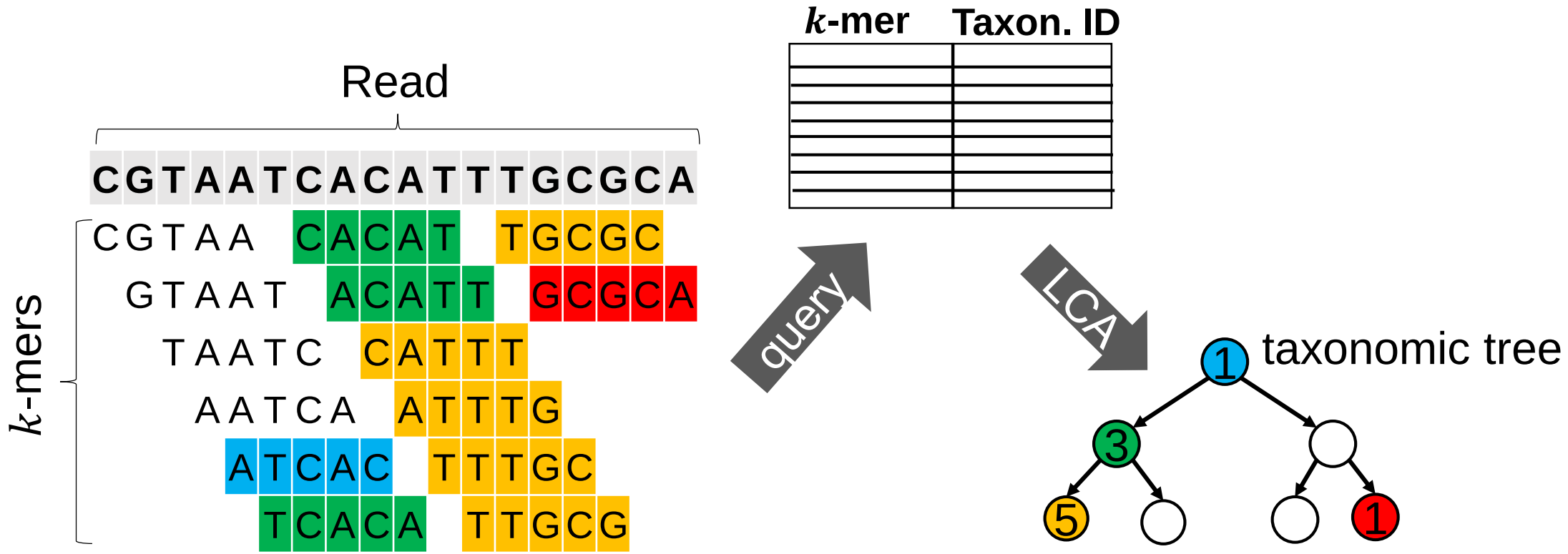



DREI MENSCHEN STARBEN NACH VERZEHR VON VERSEUCHTEM FLEISCH

Verwirrung um Tote durch Wilke-Wurst

- Collaboration with Prof. T. Hankeln's group (Biology, JGU)
  - Liu, Ripp, Köppel, Schmidt, Hellmann, Weber, Krombholz, Schmidt, Hankeln: *AFS: identification and quantification of species composition by metagenomic sequencing*. **Bioinformatics** 33(9):2017

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**

$S_s(X)$ = set of s smallest hash values $h(x)$ of all $x \in X$

3-mer hashes:

CTAGCTTAATAT $A_h = \{$ 83 229 55 198 128 184 79 57 188 165 $\}$

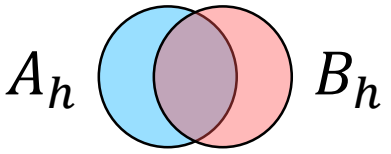
CTAGCATAATAT $B_h = \{$ 83 229 55 81 90 188 79 57 188 165 $\}$

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

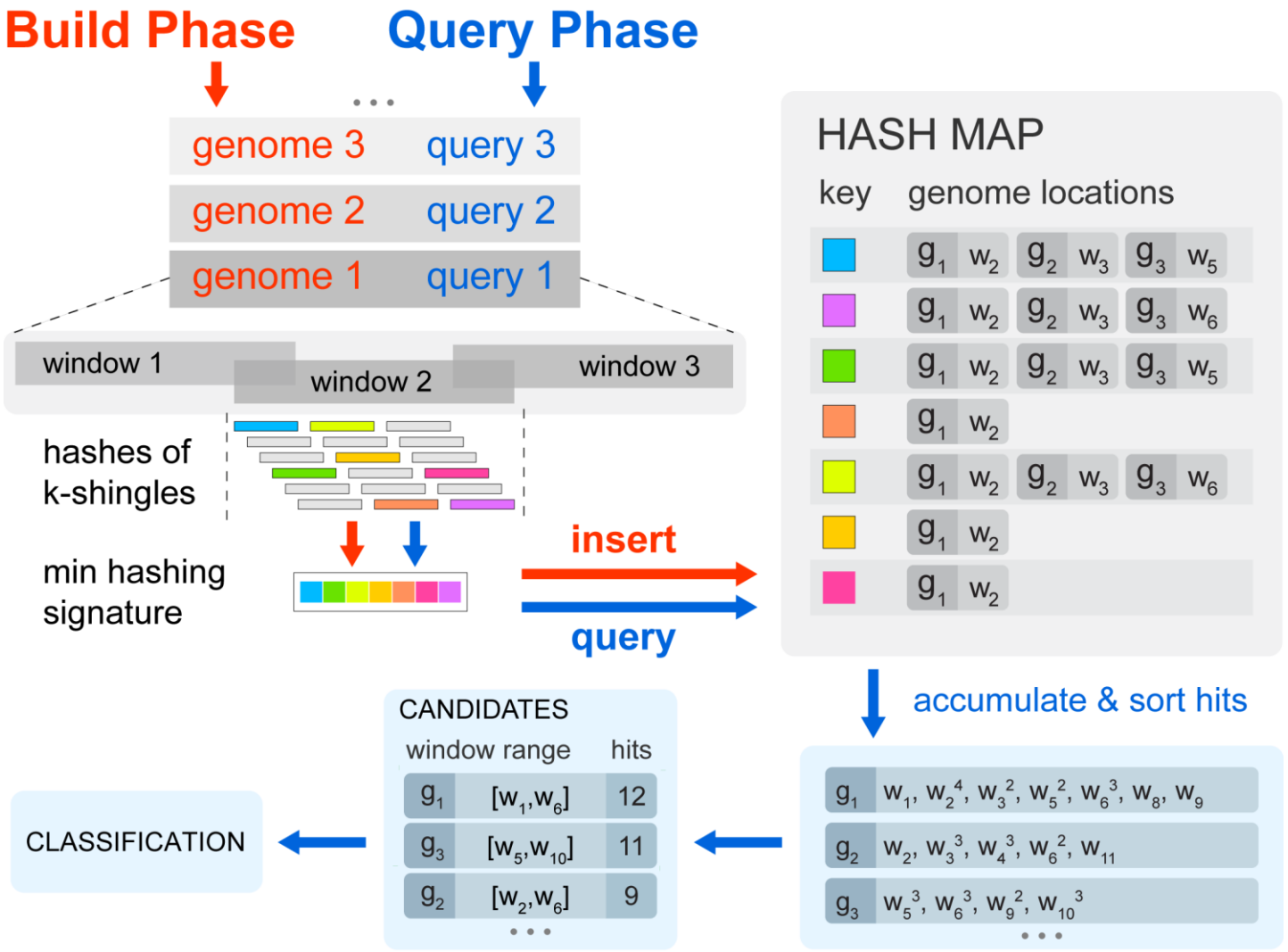
$S_4(A_h) = \{\mathbf{55}, \mathbf{57}, \mathbf{79}, 83\}$

$S_4(B_h) = \{\mathbf{55}, \mathbf{57}, \mathbf{79}, 81\}$

$$M_4(A, B) = \frac{3}{5} = 0.6$$

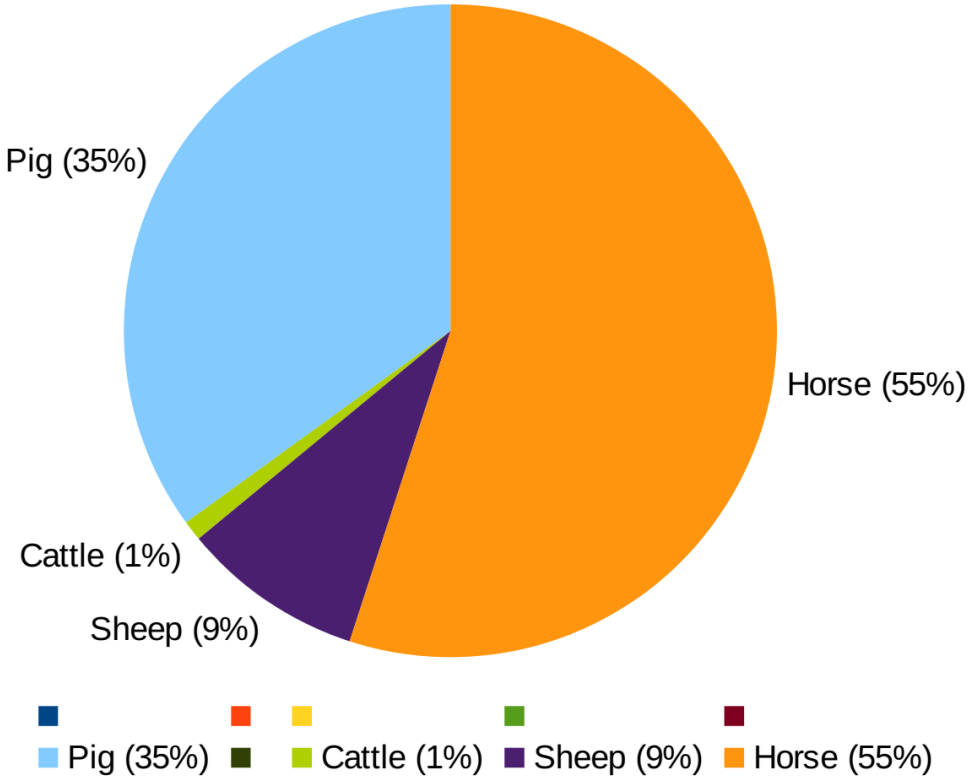


MetaCache-GPU

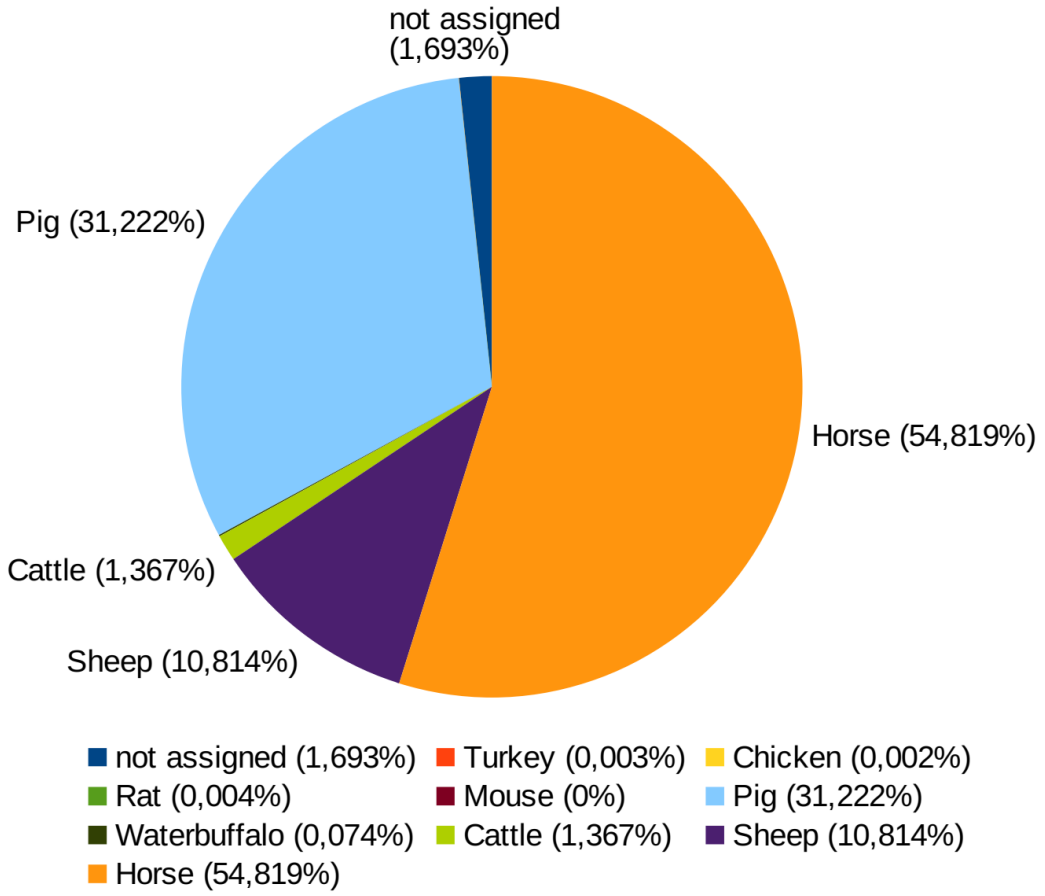


Metache-GPU: Accuracy

Ground Truth



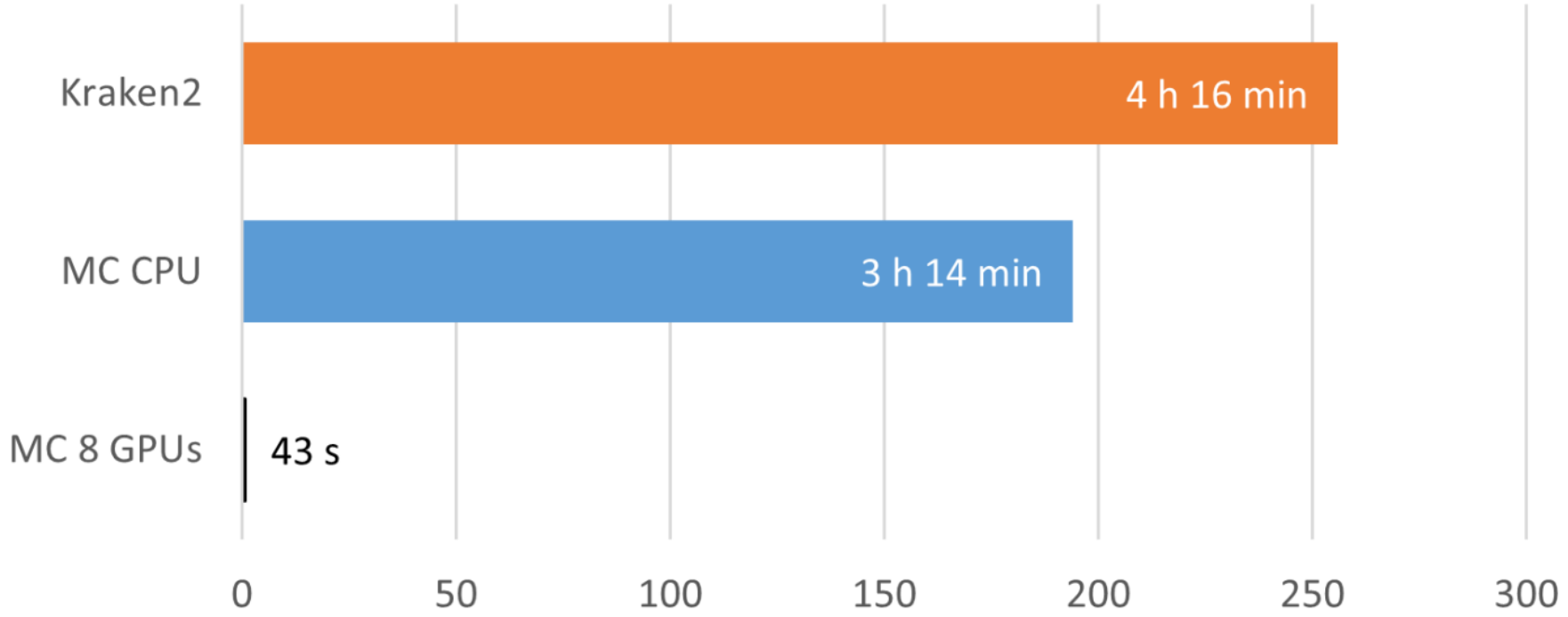
AFS-MetaCache Estimation



- Tested on sequenced “**calibrator sausages**”

MetaCache-GPU: Performance

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

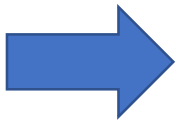


- Kobus, Müller, Jünger, Hundt, Schmidt: *MetaCache-GPU: Ultra-Fast Metagenomic Classification*, **ICPP 2021**

k-mer Counting

R = {ACGTTA, ACGTTA, ACGTTT}

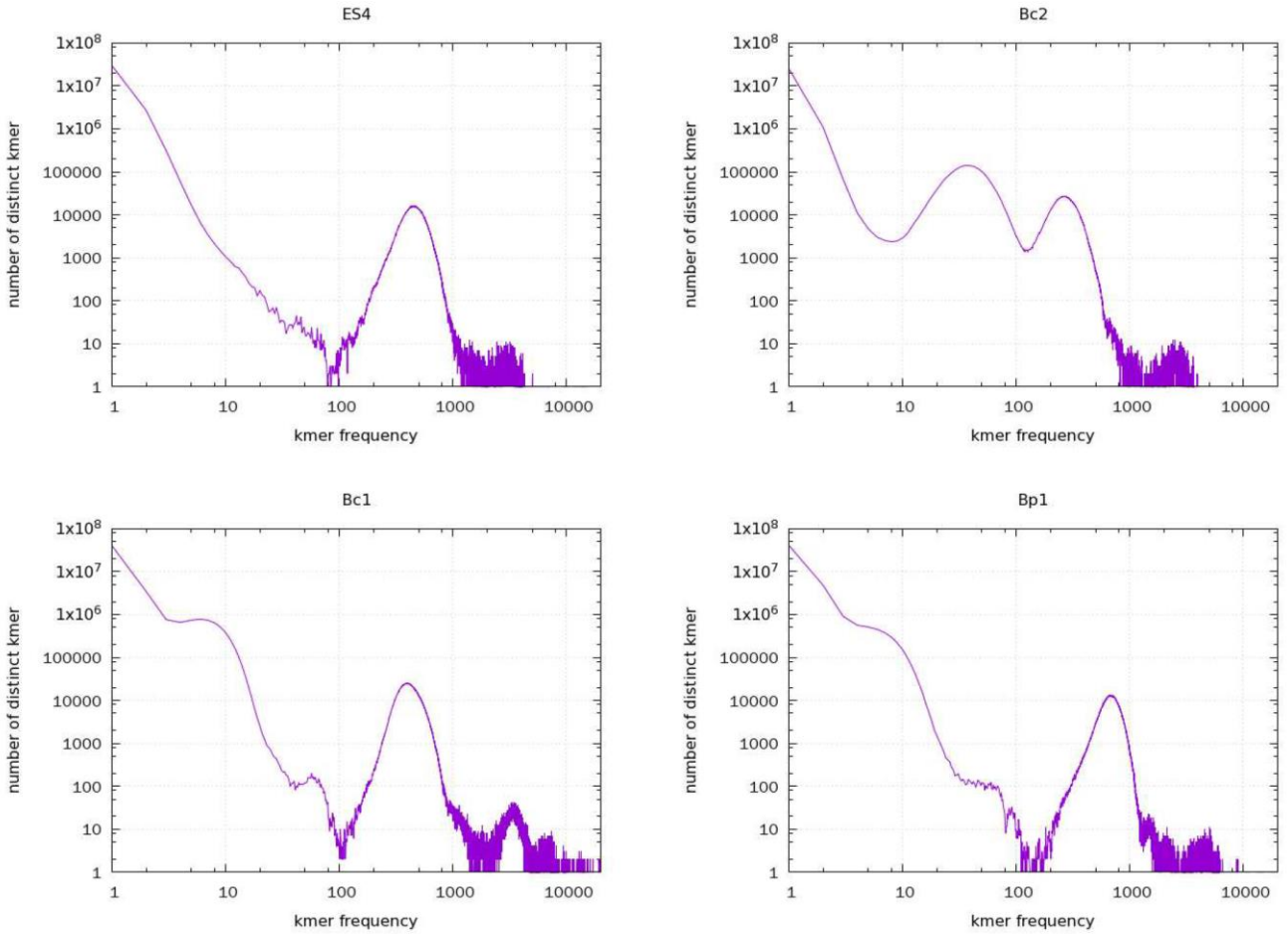
$k = 4$



k-mer	count
ACGT	3
CGTT	3
GTTA	2
GTTT	1

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

Cross-species contamination in NGS data



PeerJ

Unexpected cross-species contamination in genome sequencing projects

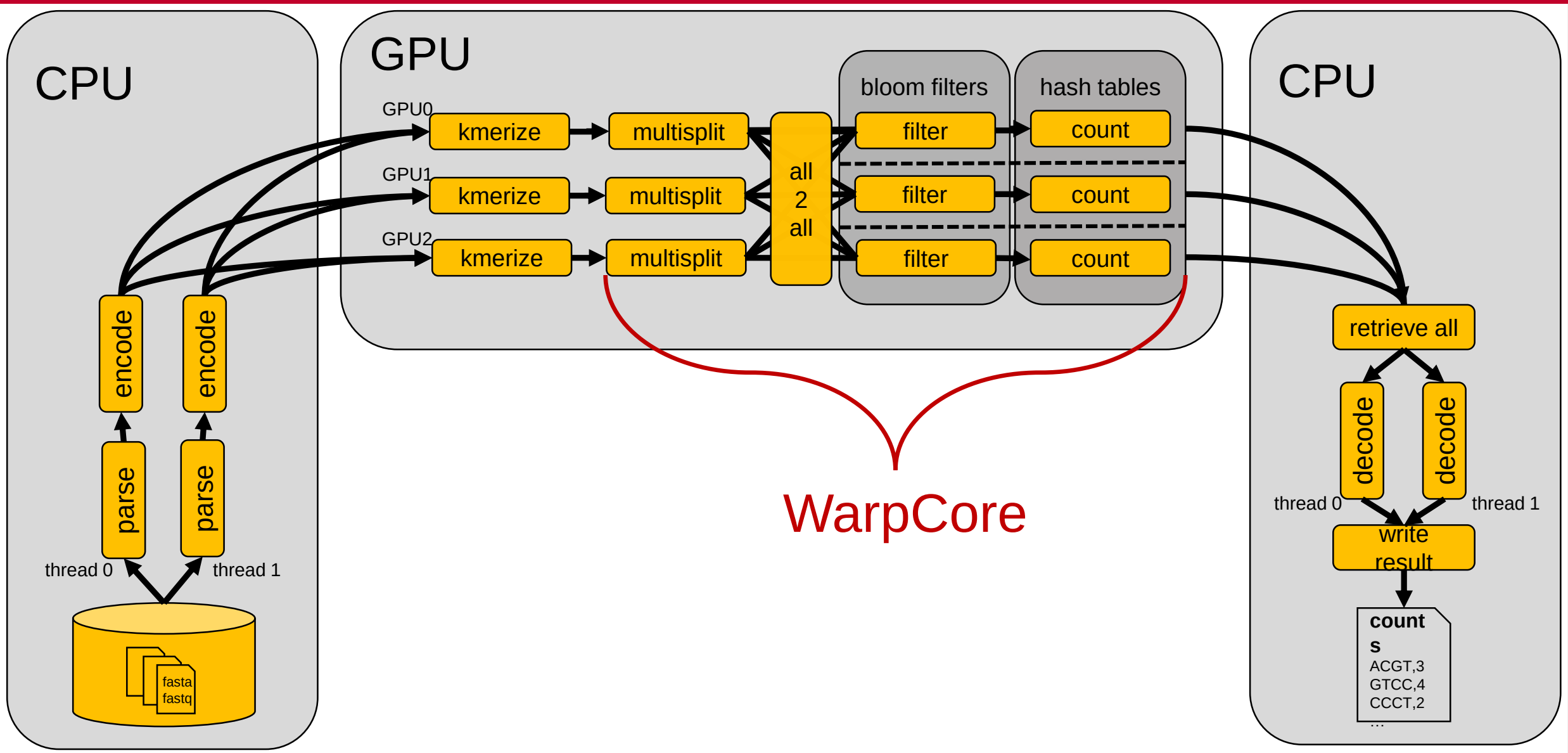
Samier Merchant[1,2], Derrick E. Wood[1,3] and Steven L. Salzberg[1,3,4]

[1] Center for Computational Biology, McKusick-Nathans Institute of Genetic Medicine, Johns Hopkins University, Baltimore, MD, USA
[2] Department of Computer Science, Brown University, Providence, RI, USA
[3] Department of Computer Science, Johns Hopkins University, USA
[4] Department of Biomedical Engineering, Johns Hopkins University, USA

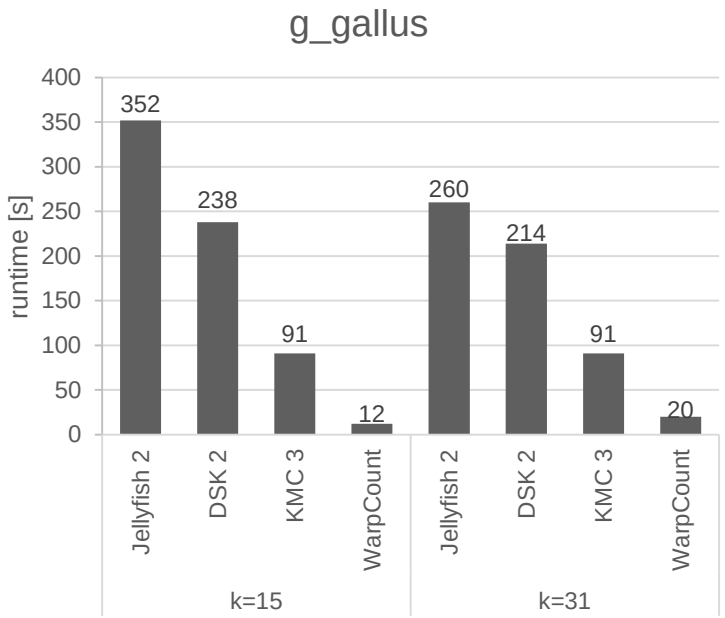
ABSTRACT

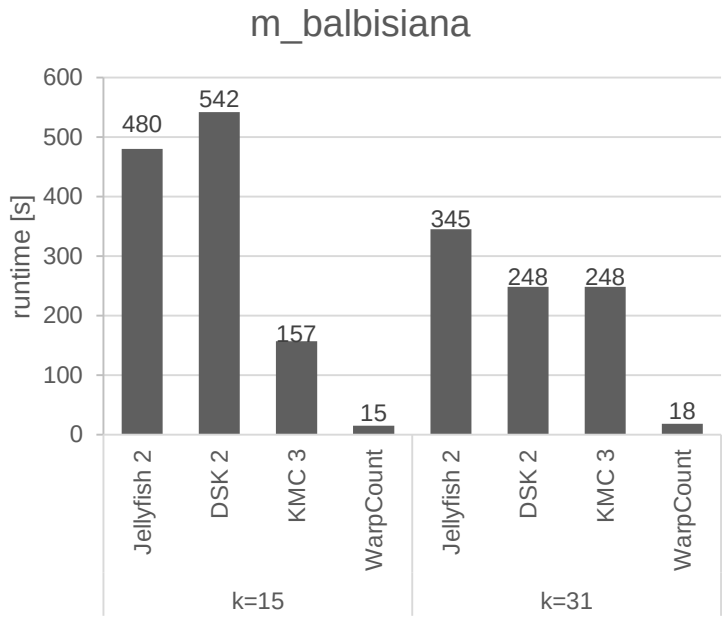
- ***k*-mer abundance histograms** could be used as indicator for contamination
- For haploid organisms: the distribution should resemble a single Gaussian + low frequency *k*-mers indicating sequencing errors

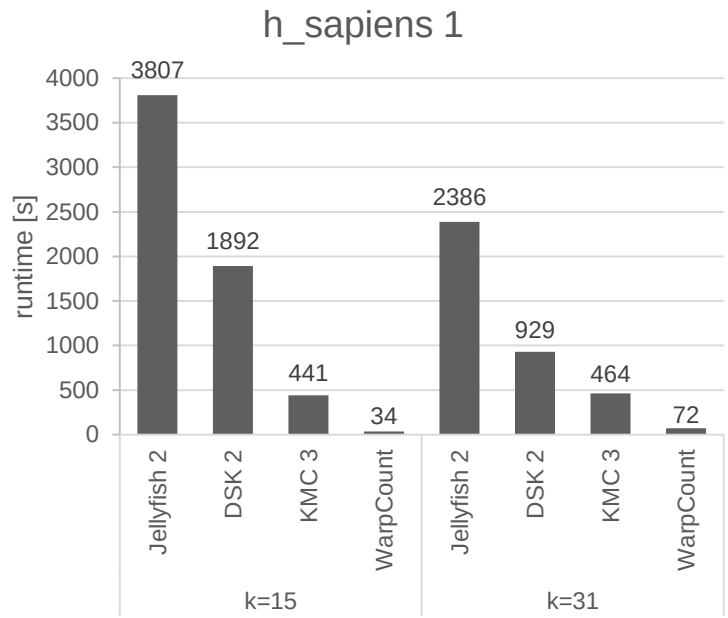
WarpCount


CPU
GPU
CPU
bloom filters
hash tables
GPU0
GPU1
GPU2
kmerize
kmerize
kmerize
multisplit
multisplit
multisplit
all
2
all
filter
filter
filter
count
count
count
encode
encode
parse
parse
thread 0
thread 1
fasta
fastq
WarpCore
retrieve all
decode
decode
thread 0
thread 1
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

Conclusions

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)