University of Stuttgart
Germany

Institute for Parallel and Distributed Systems
Scientific Computing

oneAPI Developer Summit at ISC-HPC June 22nd-23rd

Marcel Breyer

Performance-Portable Distributed k-Nearest Neighbors using Locality-Sensitive Hashing and SYCL

University of Stuttgart
Germany

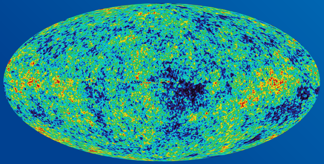
IPVS

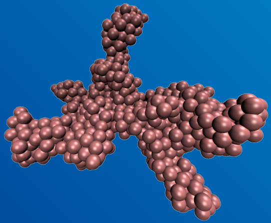
Institute for Parallel and Distributed Systems

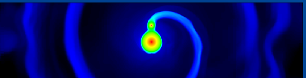
Scientific Computing

Marcel Breyer

Prof. Dr.
Dirk Pflüger







Motivation

- **k-Nearest Neighbors** used as building block in many algorithms
  → e.g. classifier in data mining (proposed by Thomas Cover and P. Hart in 1967)

Motivation

- **k-Nearest Neighbors** used as building block in many algorithms
  → e.g. classifier in data mining (proposed by Thomas Cover and P. Hart in 1967)
- naive brute-force approach is infeasible for large data sets

Motivation

- **k-Nearest Neighbors** used as building block in many algorithms
  → e.g. classifier in data mining (proposed by Thomas Cover and P. Hart in 1967)
- naive brute-force approach is infeasible for large data sets
- instead use approximate algorithms

Motivation

- **k-Nearest Neighbors** used as building block in many algorithms
  → e.g. classifier in data mining (proposed by Thomas Cover and P. Hart in 1967)
- naive brute-force approach is infeasible for large data sets
- instead use approximate algorithms

→ **Locality-Sensitive Hashing**

Motivation

- **k-Nearest Neighbors** used as building block in many algorithms
  → e.g. classifier in data mining (proposed by Thomas Cover and P. Hart in 1967)
- naive brute-force approach is infeasible for large data sets
- instead use approximate algorithms

→ **Locality-Sensitive Hashing**

- supporting even bigger data sets

Motivation

- **k-Nearest Neighbors** used as building block in many algorithms
  → e.g. classifier in data mining (proposed by Thomas Cover and P. Hart in 1967)
- naive brute-force approach is infeasible for large data sets
- instead use approximate algorithms

→ **Locality-Sensitive Hashing**

- supporting even bigger data sets

→ **multi-GPU support**

Motivation

- **k-Nearest Neighbors** used as building block in many algorithms
  → e.g. classifier in data mining (proposed by Thomas Cover and P. Hart in 1967)
- naive brute-force approach is infeasible for large data sets
- instead use approximate algorithms

→ Locality-Sensitive Hashing

- supporting even bigger data sets

→ multi-GPU support

- different GPU hardware from Intel, AMD, and NVIDIA

Motivation

- **k-Nearest Neighbors** used as building block in many algorithms
  → e.g. classifier in data mining (proposed by Thomas Cover and P. Hart in 1967)
- naive brute-force approach is infeasible for large data sets
- instead use approximate algorithms

→ Locality-Sensitive Hashing

- supporting even bigger data sets

→ multi-GPU support

- different GPU hardware from Intel, AMD, and NVIDIA

→ SYCL

Locality-Sensitive Hashing

1

Locality-Sensitive Hashing (proposed by Piotr Indyk and Rajeev Motwani)

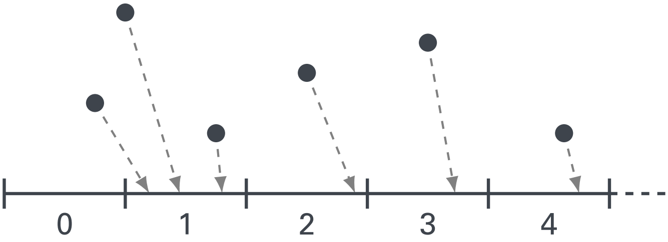
hash value	points
0	
1	
2	
3	
4	

Locality-Sensitive Hashing (proposed by Piotr Indyk and Rajeev Motwani)

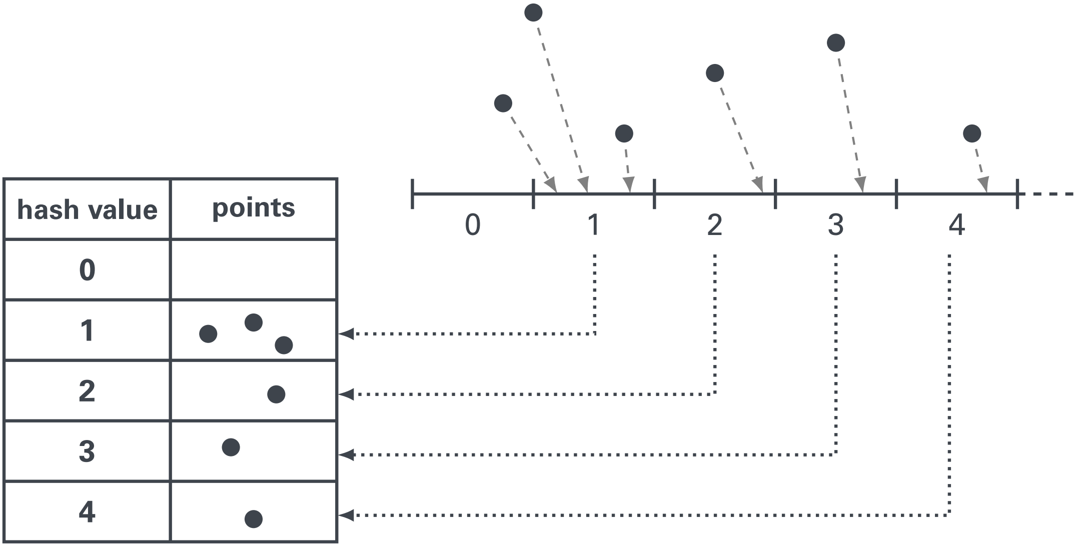
hash value	points
0	
1	
2	
3	
4	

Locality-Sensitive Hashing (proposed by Piotr Indyk and Rajeev Motwani)

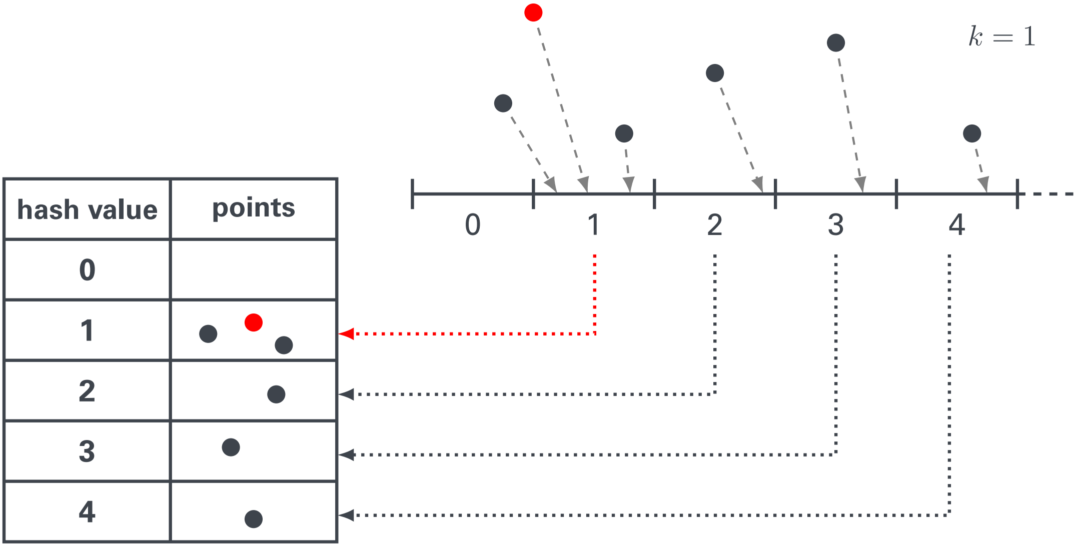
hash value	points
0	
1	
2	
3	
4	



Locality-Sensitive Hashing (proposed by Piotr Indyk and Rajeev Motwani)



Locality-Sensitive Hashing (proposed by Piotr Indyk and Rajeev Motwani)

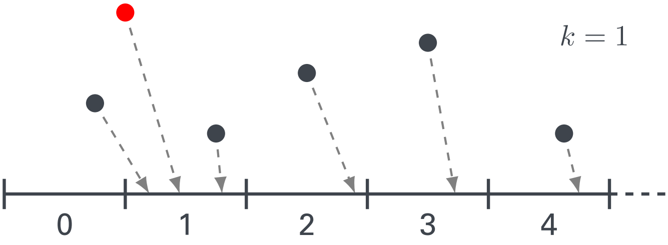


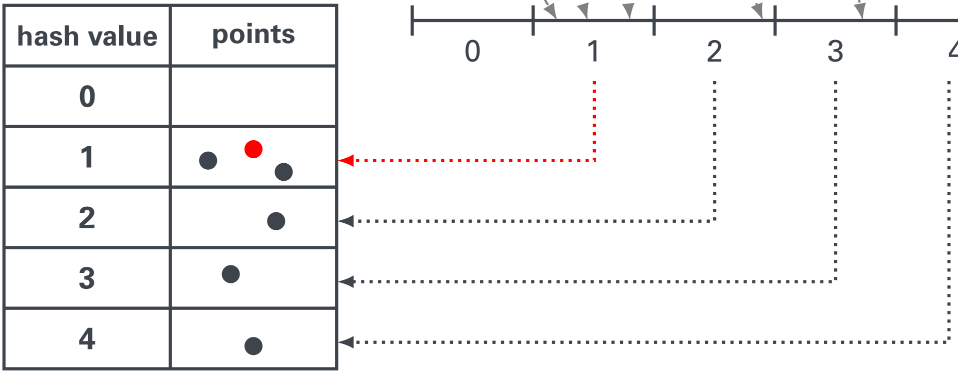
Locality-Sensitive Hashing (proposed by Piotr Indyk and Rajeev Motwani)

→ too many points per bucket

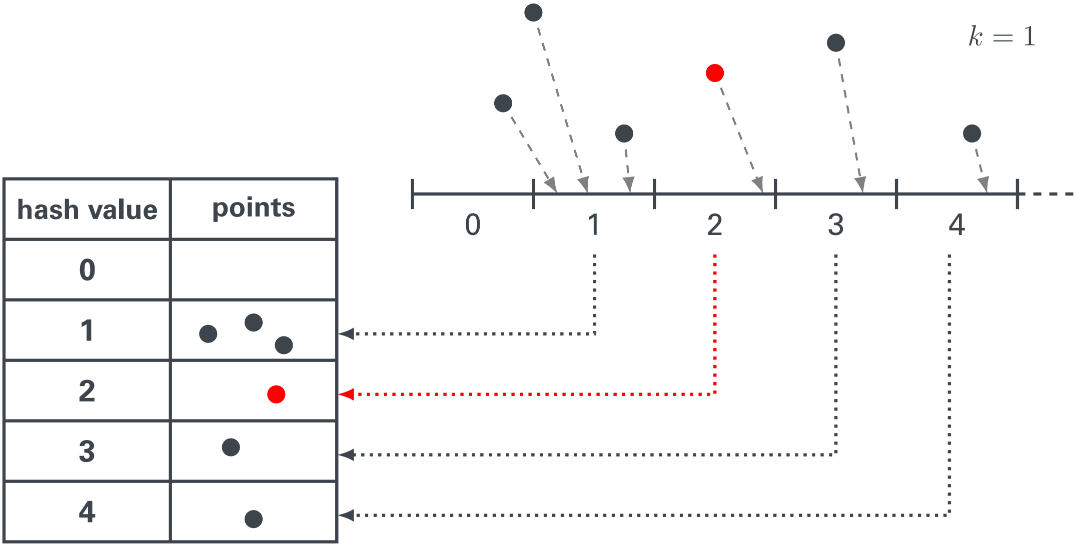
→ use multiple hash functions:

$$g(\vec{x}) = concat(h_1(\vec{x}), \ldots, h_m(\vec{x}))$$





Locality-Sensitive Hashing (proposed by Piotr Indyk and Rajeev Motwani)

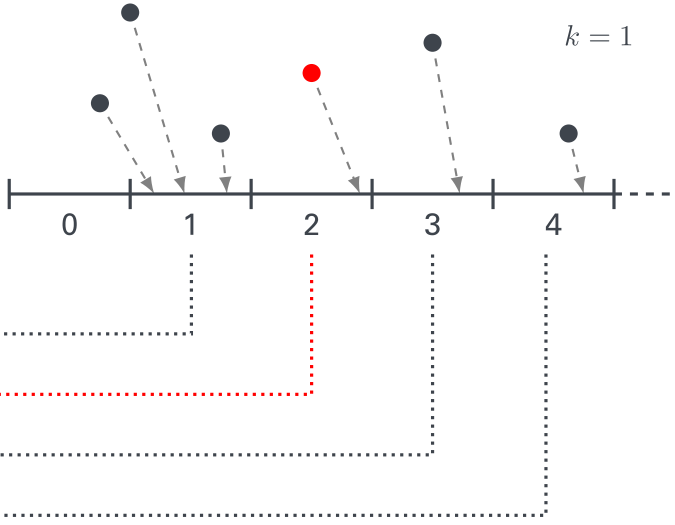


Locality-Sensitive Hashing (proposed by Piotr Indyk and Rajeev Motwani)

- → too few points per bucket
- → use multiple hash tables:

$g_0(\vec{x}), g_1(\vec{x}), \ldots, g_m(\vec{x})$

hash value	points
0	
1	
2	
3	
4	



Locality-Sensitive Hash Functions

Random Projections

(proposed by Mayur Datar et al.)

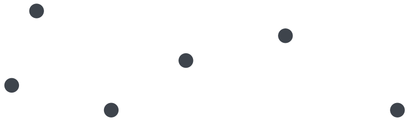
Entropy-Based Hash Functions

(proposed by Qiang Wang et al.)

Locality-Sensitive Hash Functions

Random Projections

(proposed by Mayur Datar et al.)



Entropy-Based Hash Functions

(proposed by Qiang Wang et al.)

Locality-Sensitive Hash Functions

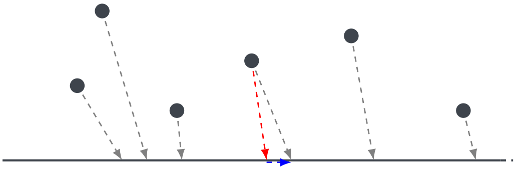
Random Projections

(proposed by Mayur Datar et al.)

Entropy-Based Hash Functions

(proposed by Qiang Wang et al.)

$$h(\vec{x}) = \frac{\vec{a} \cdot \vec{x} + b}{}$$



$\vec{a} \in \mathbb{R}^d$: independently chosen from the normal distribution

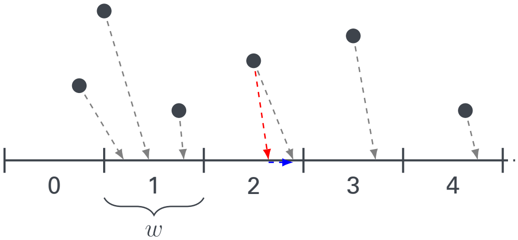
$b \in \mathbb{R}$: chosen uniformly from $[0, w]$

Locality-Sensitive Hash Functions

Random Projections

(proposed by Mayur Datar et al.)

$$h(\vec{x}) = \left\lfloor \frac{\vec{a} \cdot \vec{x} + b}{w} \right\rfloor$$



$\vec{a} \in \mathbb{R}^d$: independently chosen from the normal distribution

$b \in \mathbb{R}$: chosen uniformly from $[0, w]$

Entropy-Based Hash Functions

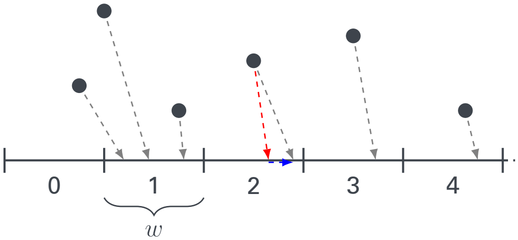
(proposed by Qiang Wang et al.)

Locality-Sensitive Hash Functions

Random Projections

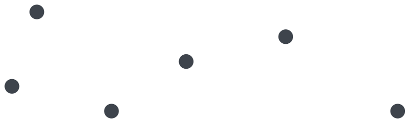
(proposed by Mayur Datar et al.)

$$h(\vec{x}) = \left\lfloor \frac{\vec{a} \cdot \vec{x} + b}{w} \right\rfloor$$



Entropy-Based Hash Functions

(proposed by Qiang Wang et al.)



$\vec{a} \in \mathbb{R}^d$: independently chosen from the normal distribution

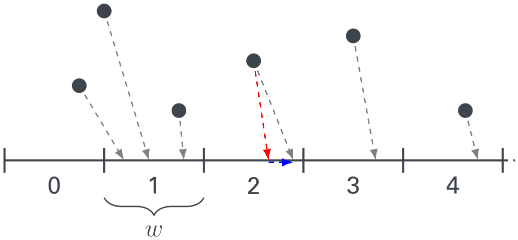
$b \in \mathbb{R}$: chosen uniformly from $[0, w]$

Locality-Sensitive Hash Functions

Random Projections

(proposed by Mayur Datar et al.)

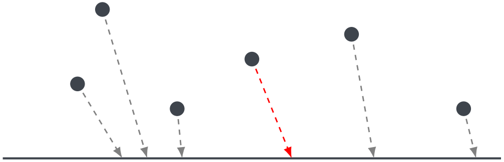
$$h(\vec{x}) = \left\lfloor \frac{\vec{a} \cdot \vec{x} + b}{w} \right\rfloor$$



Entropy-Based Hash Functions

(proposed by Qiang Wang et al.)

$$h'(\vec{x}) = \vec{a} \cdot \vec{x}$$



$\vec{a} \in \mathbb{R}^d$: independently chosen from the normal distribution

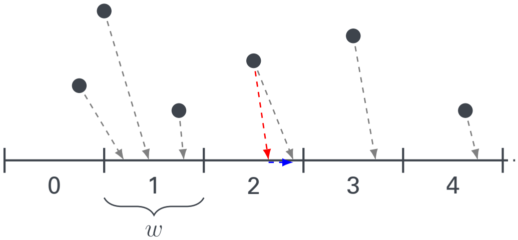
$b \in \mathbb{R}$: chosen uniformly from $[0, w]$

Locality-Sensitive Hash Functions

Random Projections

(proposed by Mayur Datar et al.)

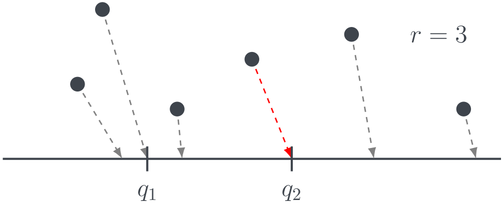
$$h(\vec{x}) = \left\lfloor \frac{\vec{a} \cdot \vec{x} + b}{w} \right\rfloor$$



Entropy-Based Hash Functions

(proposed by Qiang Wang et al.)

$$h'(\vec{x}) = \vec{a} \cdot \vec{x}$$



$\vec{a} \in \mathbb{R}^d$: independently chosen from the normal distribution

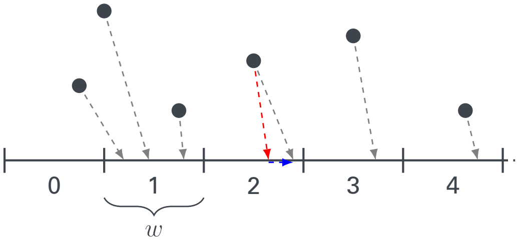
$b \in \mathbb{R}$: chosen uniformly from $[0, w]$

Locality-Sensitive Hash Functions

Random Projections

(proposed by Mayur Datar et al.)

$$h(\vec{x}) = \left\lfloor \frac{\color{red}{\vec{a}} \cdot \vec{x} + \color{blue}{b}}{w} \right\rfloor$$

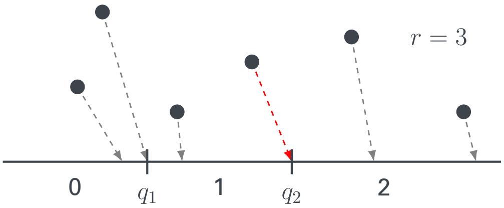


Entropy-Based Hash Functions

(proposed by Qiang Wang et al.)

$$h'(\vec{x}) = \color{red}{\vec{a}} \cdot \vec{x}$$

$$h(\vec{x}) = \begin{cases} 0 & h'(\vec{x}) \leq q_1 \\ 1 & q_1 < h'(\vec{x}) \leq q_2 \\ 2 & h'(\vec{x}) > q_2 \end{cases}$$



$\color{red}{\vec{a}} \in \mathbb{R}^d$: independently chosen from the normal distribution

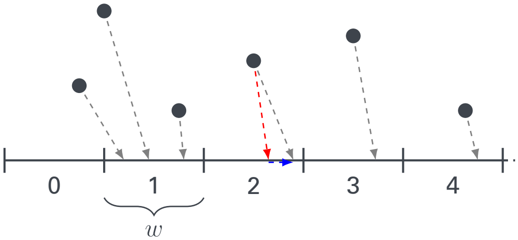
$\color{blue}{b} \in \mathbb{R}$: chosen uniformly from $[0, w]$

Locality-Sensitive Hash Functions

Random Projections

(proposed by Mayur Datar et al.)

$$h(\vec{x}) = \left\lfloor \frac{\vec{a} \cdot \vec{x} + b}{w} \right\rfloor$$



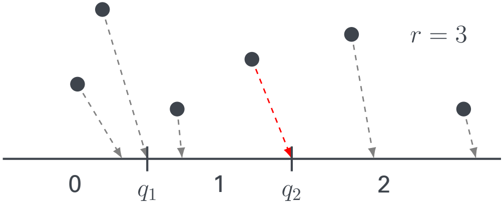
- \+ efficient to create
- \+ $h(\vec{x})$ efficient to calculate
- − non-uniform distribution over segments

Entropy-Based Hash Functions

(proposed by Qiang Wang et al.)

$$h'(\vec{x}) = \vec{a} \cdot \vec{x}$$

$$h(\vec{x}) = \begin{cases} 0 & h'(\vec{x}) \leq q_1 \\ 1 & q_1 < h'(\vec{x}) \leq q_2 \\ 2 & h'(\vec{x}) > q_2 \end{cases}$$

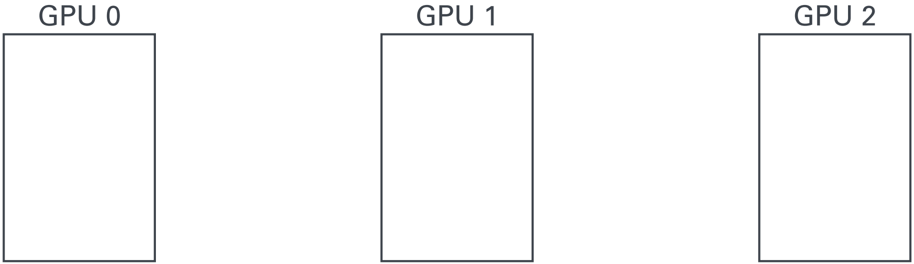


- − inefficient to create
- \+ $h(\vec{x})$ efficient to calculate
- \+ uniform distribution over segments

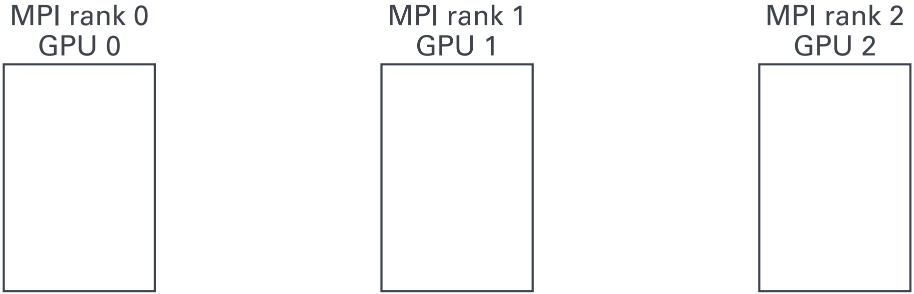
Implementation

2

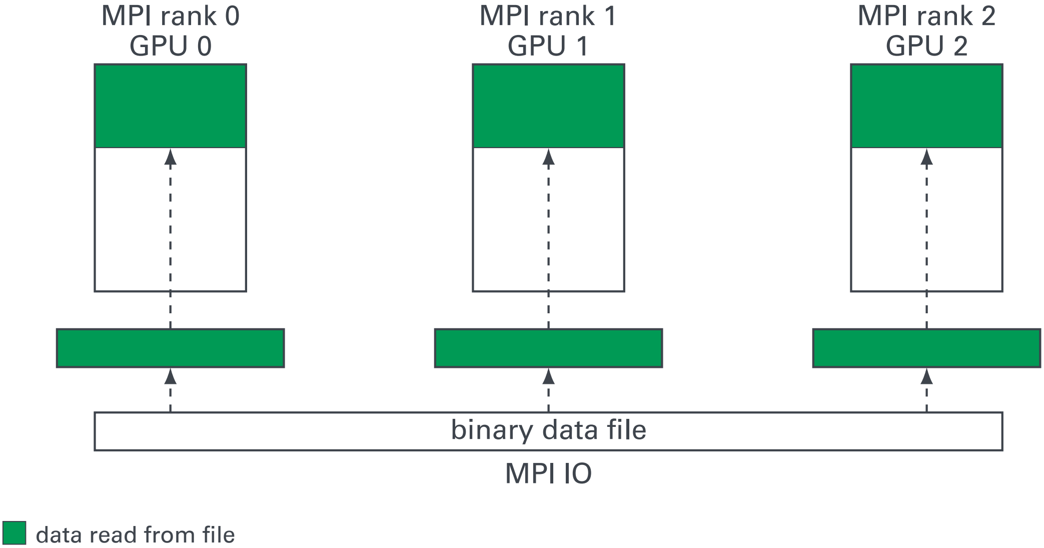
Distributed Multi-GPU Support using MPI



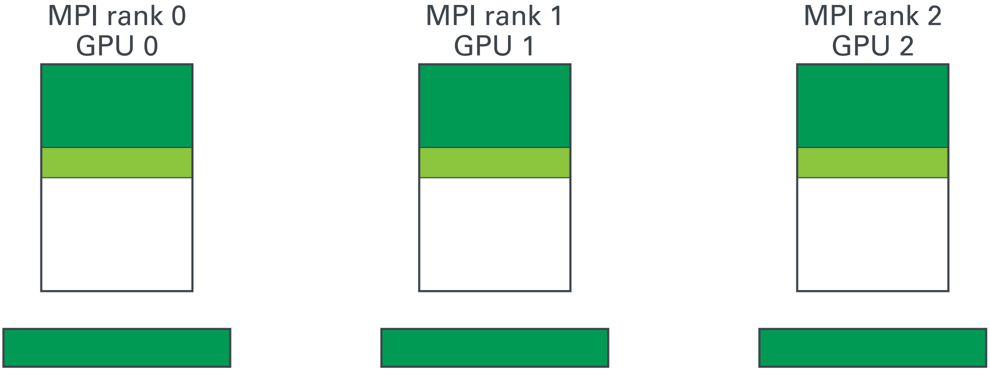
Distributed Multi-GPU Support using MPI



Distributed Multi-GPU Support using MPI

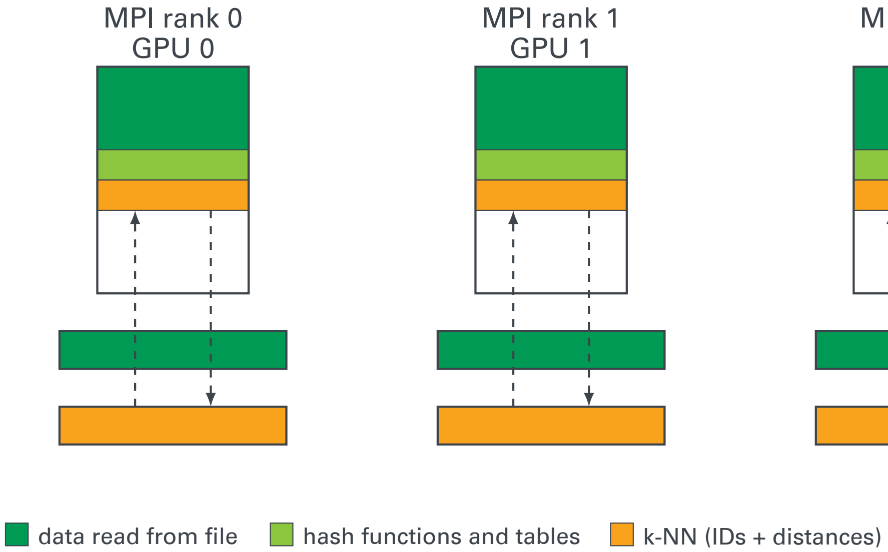


Distributed Multi-GPU Support using MPI

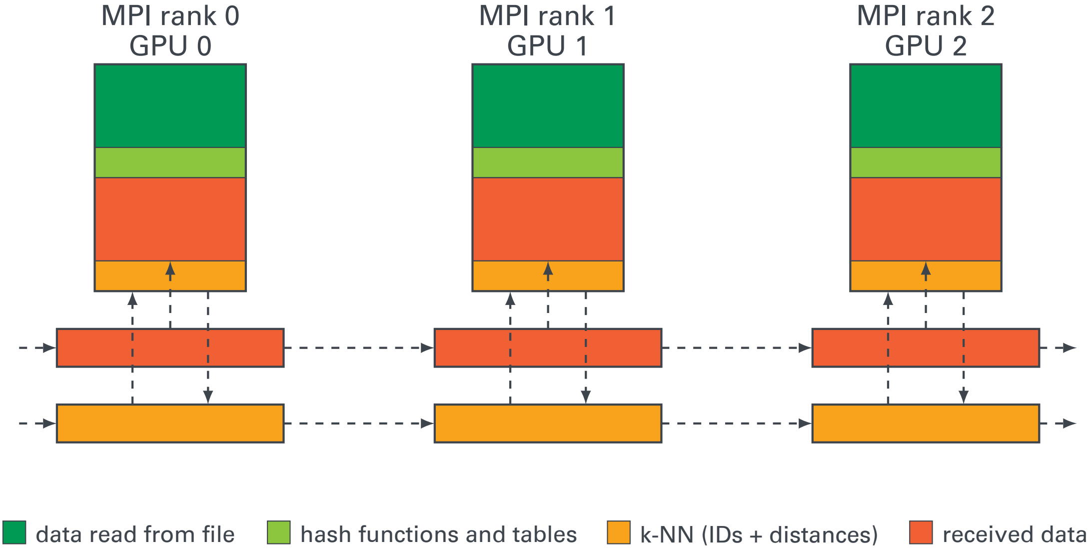


data read from file   hash functions and tables

Distributed Multi-GPU Support using MPI



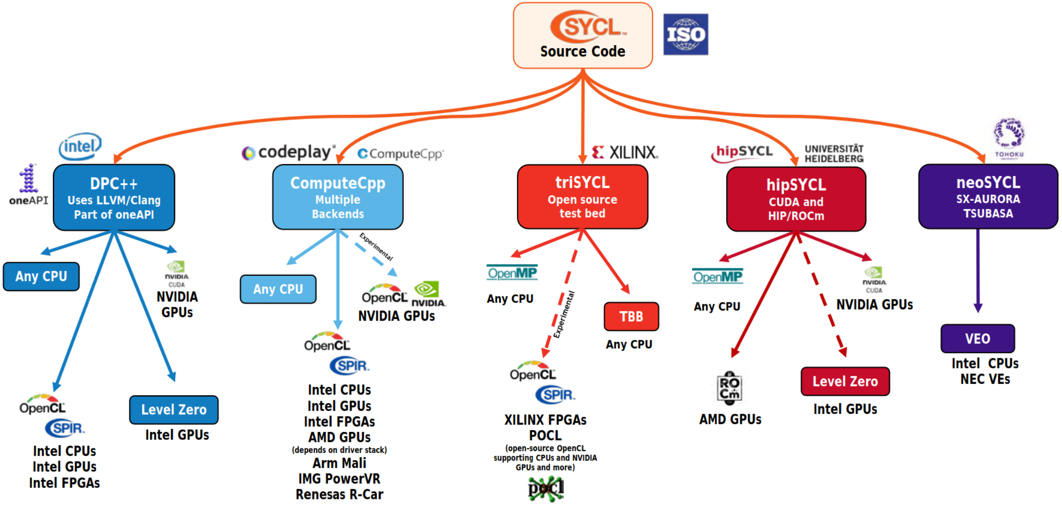
Distributed Multi-GPU Support using MPI



SYCL

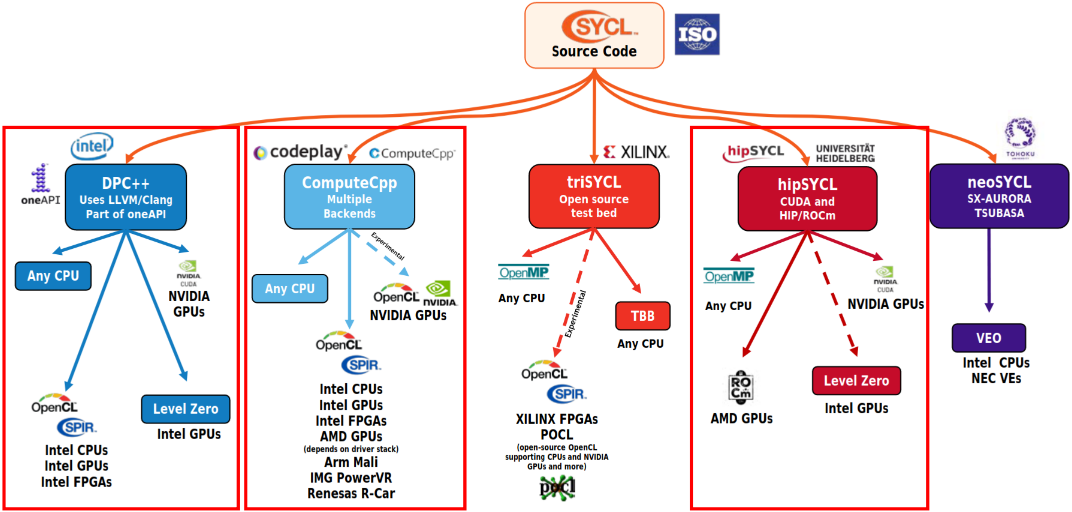
3

SYCL Implementations



Based on a slide by Michael Wong: `https://www.iwocl.org/wp-content/uploads/k04-iwocl-syclcon-2021-wong-slides.pdf` (27.05.2021)

SYCL Implementations



Based on a slide by Michael Wong: `https://www.iwocl.org/wp-content/uploads/k04-iwocl-syclcon-2021-wong-slides.pdf` (27.05.2021)

Results

4

Setup

Intel DevCloud	local system 1	local system 2	local system 3
Intel i9-10920X @ 3.5 GHz	Intel i9-10980XE @ 3.0 GHz	AMD EPYC 7551P @ 2.0 GHz	Intel Xeon Gold 5120 @ 2.2 GHz
Intel Iris X^e MAX	NVIDIA RTX 3080	AMD Radeon VII (VEGA 20)	8x NVIDIA GTX 1080 Ti
DPC++,[1] ComputeCpp,[3] hipSYCL[4]	DPC++,[2] ComputeCpp,[3] hipSYCL[4]	hipSYCL[4]	DPC++,[2] ComputeCpp,[3] hipSYCL[4]

[1] intel-llvm `sycl` branch (`bddb95108326`) [2] intel-llvm `sycl` branch (`7e4a38606069`) [3] v2.5.0 [4] v0.9.1

Setup

Intel DevCloud	local system 1	local system 2	local system 3
Intel i9-10920X @ 3.5 GHz	Intel i9-10980XE @ 3.0 GHz	AMD EPYC 7551P @ 2.0 GHz	Intel Xeon Gold 5120 @ 2.2 GHz
Intel Iris X^e MAX	NVIDIA RTX 3080	AMD Radeon VII (VEGA 20)	8x NVIDIA GTX 1080 Ti
DPC++,[1] ComputeCpp,[3] hipSYCL[4]	DPC++,[2] ComputeCpp,[3] hipSYCL[4]	hipSYCL[4]	DPC++,[2] ComputeCpp,[3] hipSYCL[4]

[1] intel-llvm sycl branch (bddb95108326) [2] intel-llvm sycl branch (7e4a38606069) [3] v2.5.0 [4] v0.9.1

friedman: 500 000 points in 10 dimensions (synthetic)

HIGGS: 1 000 000 points in 27 dimensions (real world)

Evaluation Metrics

$$\frac{\text{true positives}}{\text{relevant elements}}$$

recall

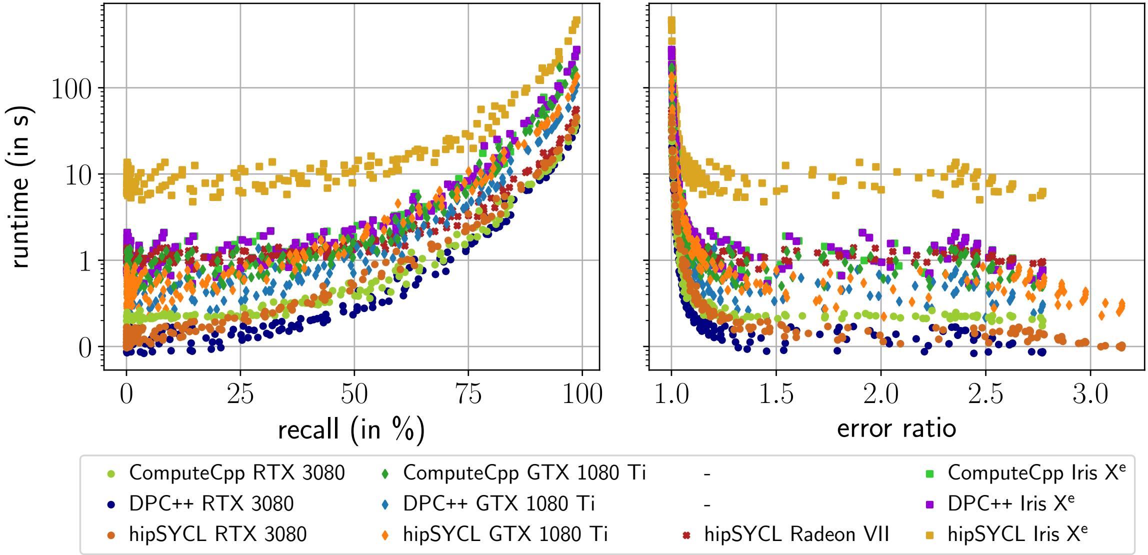
$$\frac{1}{N} \cdot \sum_{i=1}^{N} (\frac{1}{k} \cdot \sum_{j=1}^{k} \frac{dist_{LSH_j}}{dist_{correct_j}})$$

error ratio

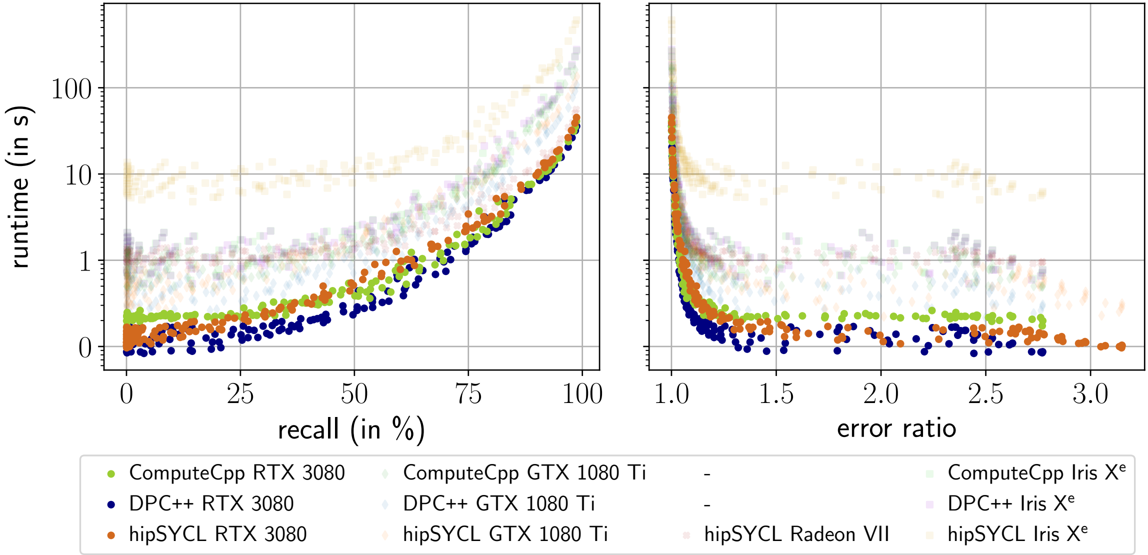
$$S_p = \frac{T_1}{T_p}$$

speedup

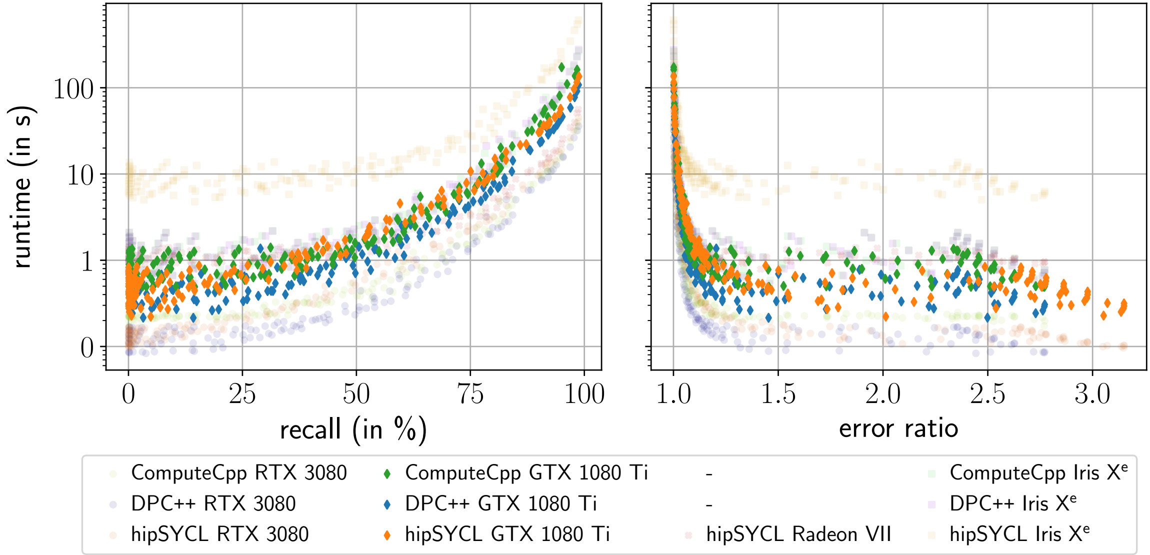
Random Projections - `friedman`



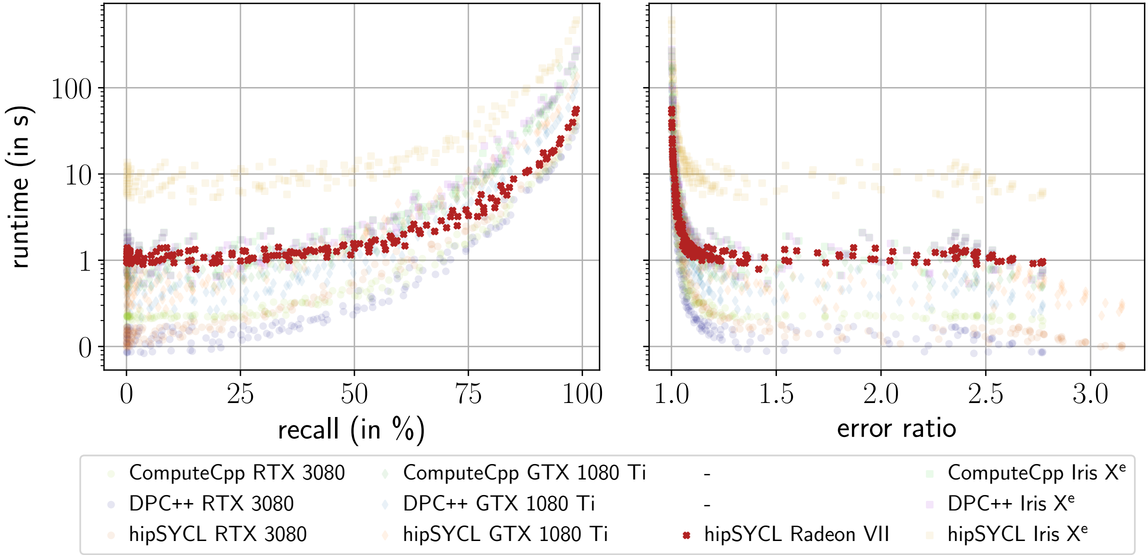
Random Projections - `friedman`



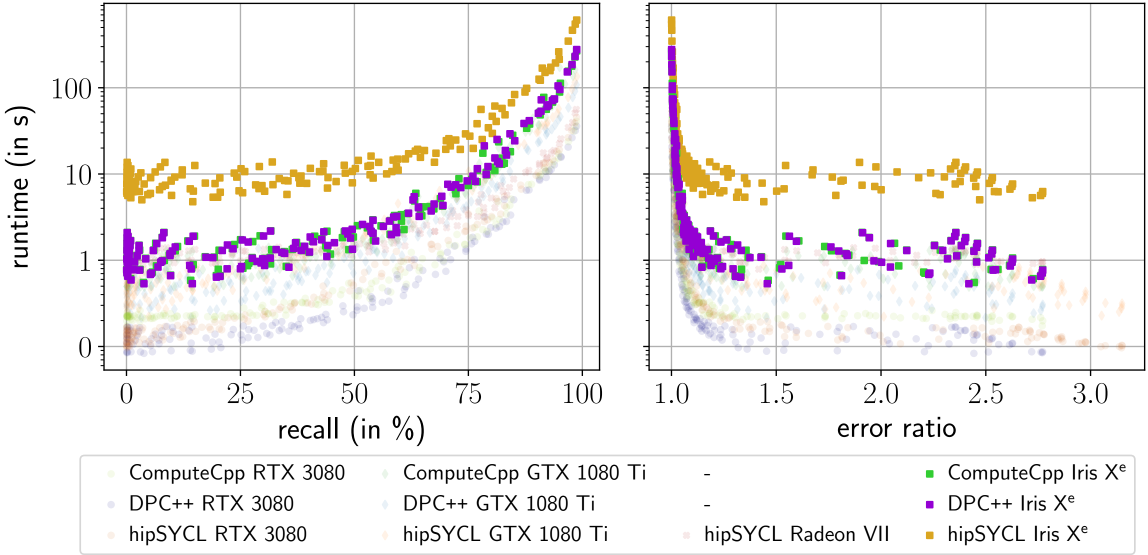
Random Projections - `friedman`



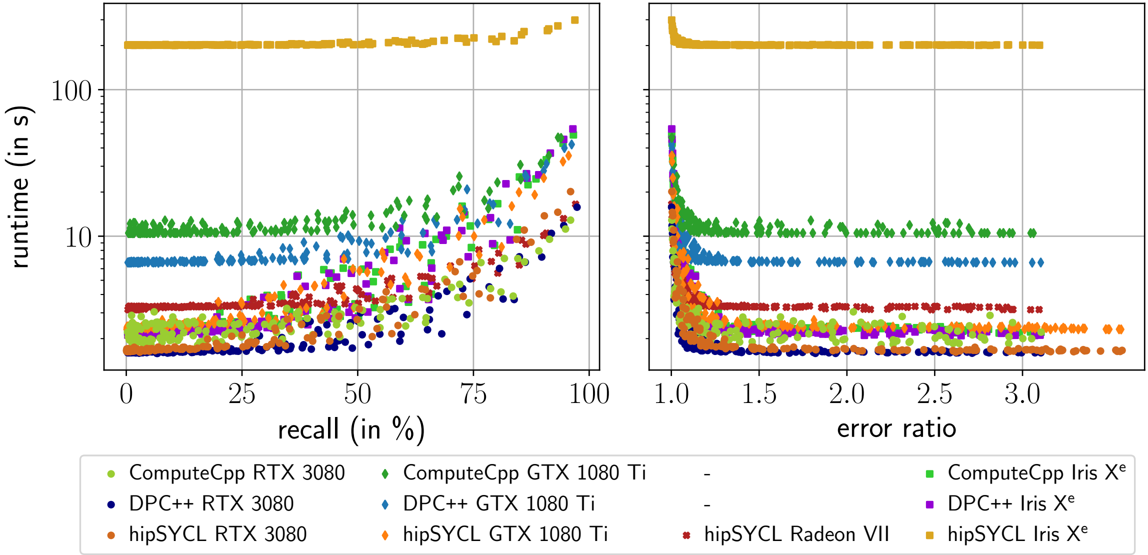
Random Projections - `friedman`



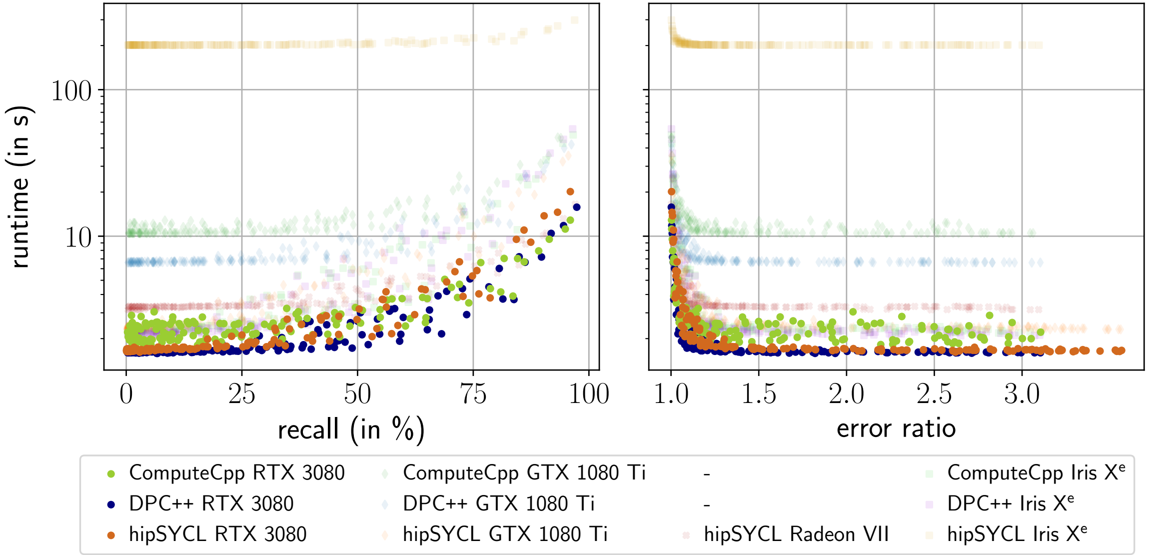
Random Projections - `friedman`



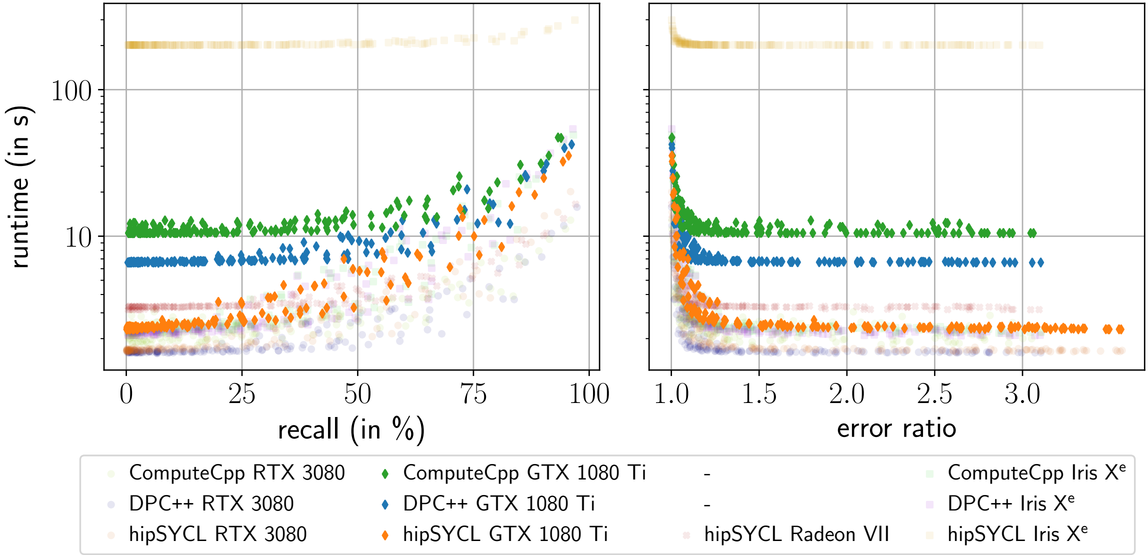
Entropy-Based Hash Functions - `friedman`



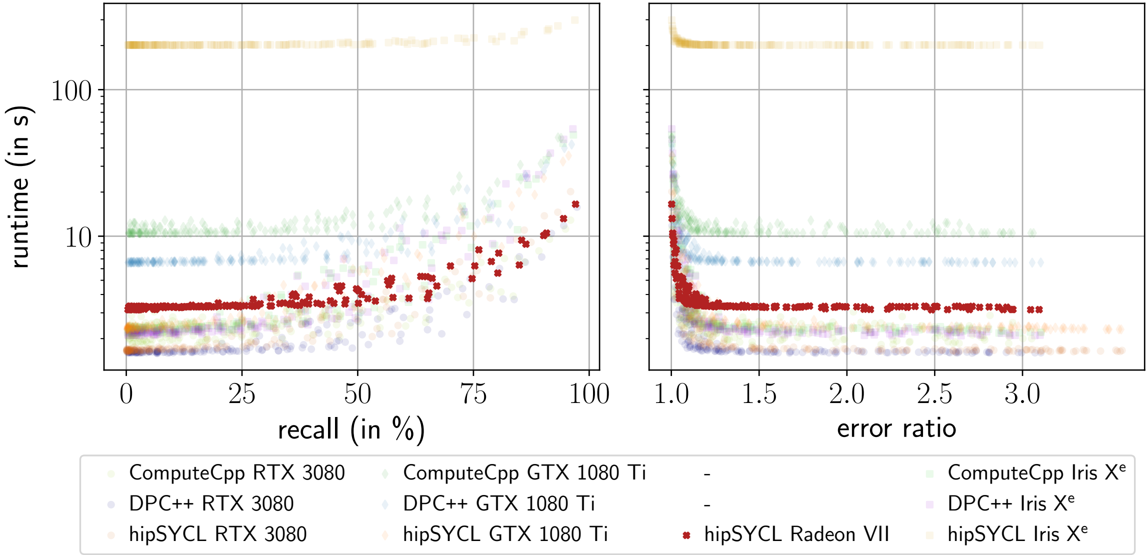
Entropy-Based Hash Functions - `friedman`



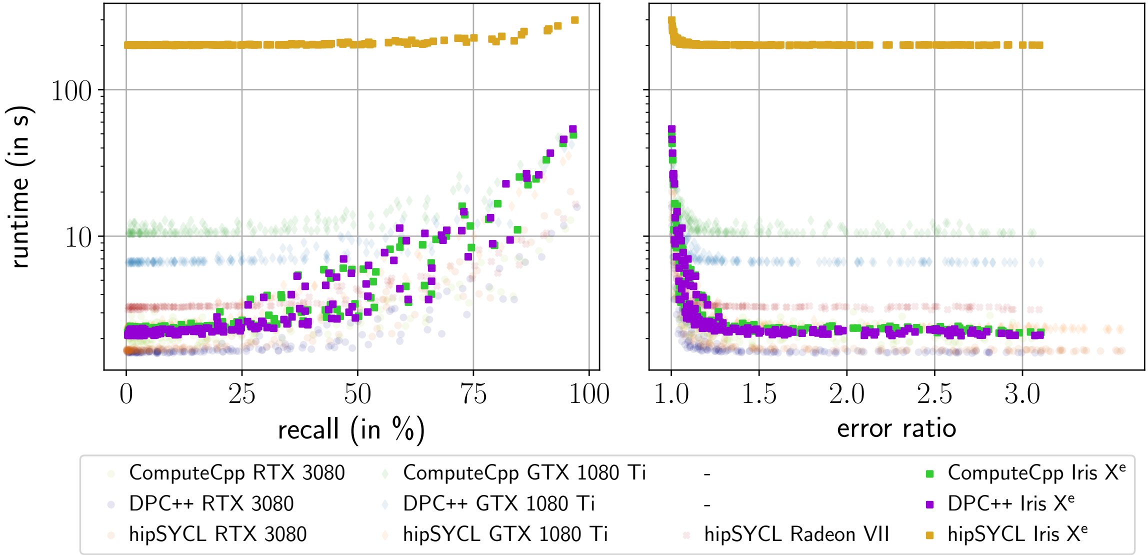
Entropy-Based Hash Functions - `friedman`



Entropy-Based Hash Functions - `friedman`



Entropy-Based Hash Functions - `friedman`



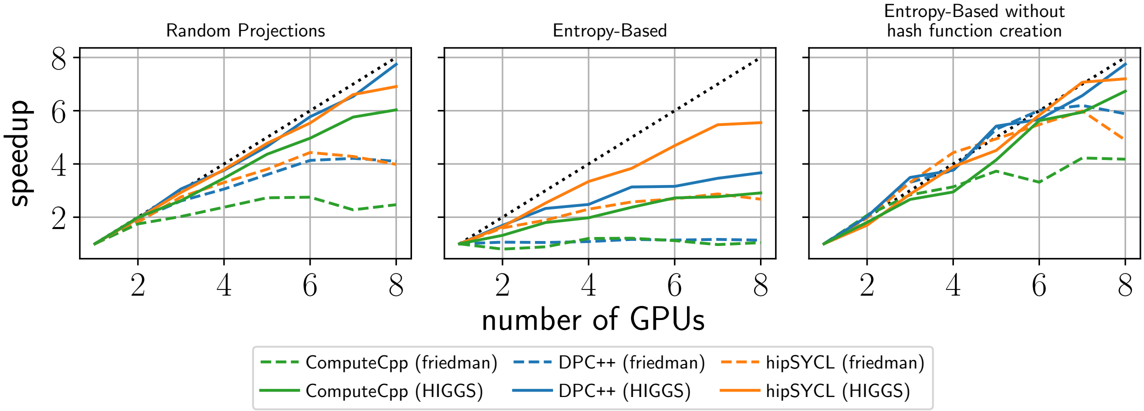
Overview

Random Projections	ComputeCpp	DPC++	hipSYCL
NVIDIA RTX 3080	✓	✓	✓
NVIDIA GTX 1080 Ti	✓	✓	✓
AMD Radeon VII	✗	✗	✓
Intel Iris X^e	✓	✓	✓

Entropy-Based Hash Functions	ComputeCpp	DPC++	hipSYCL
NVIDIA RTX 3080	✓	✓	✓
NVIDIA GTX 1080 Ti	✓	✓	✓
AMD Radeon VII	✗	✗	✓
Intel Iris X^e	✓	✓	✓

Scaling - Speedup on up to 8 NVIDIA GTX 1080 Ti

	Random Projections	Entropy-Based	Entropy-Based without hash function creation
HIGGS:	88.9 s → 14.7 s	77.9 s → 26.8 s	62.9 s → 9.3 s
	37.6 s → 4.9 s	71.9 s → 19.6 s	62.7 s → 8.1 s
	47.6 s → 6.9 s	54.0 s → 9.7 s	49.5 s → 6.9 s



Conclu-
sion

5

Conclusion

Contribution:

- Open Source implementation of **Locality-Sensitive Hashing**, an approximate k-Nearest Neighbors algorithm
  → `https://github.com/SC-SGS/Distributed_GPU_LSH_using_SYCL`

Conclusion

Contribution:

- Open Source implementation of **Locality-Sensitive Hashing**, an approximate k-Nearest Neighbors algorithm
  → `https://github.com/SC-SGS/Distributed_GPU_LSH_using_SYCL`
- distributed **multi-GPU support**

Conclusion

Contribution:

- Open Source implementation of **Locality-Sensitive Hashing**, an approximate k-Nearest Neighbors algorithm
  → `https://github.com/SC-SGS/Distributed_GPU_LSH_using_SYCL`
- distributed **multi-GPU support**
- comparing 3 different **SYCL** implementations—ComputeCpp, DPC++, and hipSYCL—on 4 different hardware platforms

Conclusion

Contribution:

- Open Source implementation of **Locality-Sensitive Hashing**, an approximate k-Nearest Neighbors algorithm
  → `https://github.com/SC-SGS/Distributed_GPU_LSH_using_SYCL`
- distributed **multi-GPU support**
- comparing 3 different **SYCL** implementations—ComputeCpp, DPC++, and hipSYCL—on 4 different hardware platforms

Results:

- comparable results for random projections and entropy-based hash functions

Conclusion

Contribution:

- Open Source implementation of **Locality-Sensitive Hashing**, an approximate k-Nearest Neighbors algorithm
  → `https://github.com/SC-SGS/Distributed_GPU_LSH_using_SYCL`
- distributed **multi-GPU support**
- comparing 3 different **SYCL** implementations—ComputeCpp, DPC++, and hipSYCL—on 4 different hardware platforms

Results:

- comparable results for random projections and entropy-based hash functions
- obtained near perfect speedup on up to 8 GPUs

Conclusion

Contribution:

- Open Source implementation of **Locality-Sensitive Hashing**, an approximate k-Nearest Neighbors algorithm
  → `https://github.com/SC-SGS/Distributed_GPU_LSH_using_SYCL`
- distributed **multi-GPU support**
- comparing 3 different **SYCL** implementations—ComputeCpp, DPC++, and hipSYCL—on 4 different hardware platforms

Results:

- comparable results for random projections and entropy-based hash functions
- obtained near perfect speedup on up to 8 GPUs
- overall similar runtime characteristics for ComputeCpp, DPC++, and hipSYCL

University of Stuttgart
Germany

IPVS

Marcel Breyer
Institute for Parallel and Distributed Systems
Scientific Computing

marcel.breyer@ipvs.uni-stuttgart.de
+49 7 11 6 85-8 84 27
https://www.ipvs.uni-stuttgart.de/institute/team/Breyer/
https://github.com/SC-SGS/Distributed_GPU_LSH_using_SYCL
https://orcid.org/0000-0003-3574-0650

Special thanks are due to M.Sc. Gregor Daiß and Prof. Dr. Dirk Pflüger.

Further Reading

Paper related to this talk

Marcel Breyer, Gregor Daiß, and Dirk Pflüger. "Performance-Portable Distributed k-Nearest Neighbors Using Locality-Sensitive Hashing and SYCL". In: IWOCL'21. 2021

Locality-Sensitive Hashing

Piotr Indyk and Rajeev Motwani. "Approximate nearest neighbors: towards removing the curse of dimensionality". In: *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 1998, pp. 604–613

Random Projections

Mayur Datar et al. "Locality-sensitive hashing scheme based on p-stable distributions". In: *Proceedings of the twentieth annual ACM symposium on Computational geometry*. ACM Press, 2004

Entropy-Based Hash Functions

Qiang Wang et al. "Entropy based locality sensitive hashing". In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2012

SYCL (DPC++)

James Reinders et al. *Data Parallel C++: Mastering DPC++ for Programming of Heterogeneous Systems using C++ and SYCL*. Springer Nature, 2021