INTEL® HPC DEVELOPER CONFERENCE

FUEL YOUR INSIGHT

INTEL® HPC DEVELOPER CONFERENCE

FUEL YOUR INSIGHT

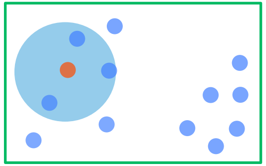
Massively Parallel K-Nearest Neighbor Computation on Distributed Architectures

Mostofa Patwary[1], Nadathur Satish[1], Narayanan Sundaram[1], Jilalin Liu[2], Peter Sadowski[2], Evan Racah[2], Suren Byna[2], Craig Tull[2], Wahid Bhimji[2] Prabhat[2], Pradeep Dubey[1]

[1]Intel Corporation, [2]Lawrence Berkeley National Lab

KNN: K-Nearest Neighbors

- Problem: Given a set of multi-dimensional data points, find the k closest neighbors



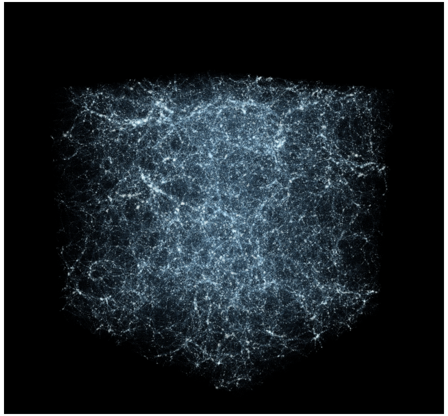
k = 3

- Classification => take the majority vote from the neighbors
- Regression => take the average value of the neighbors

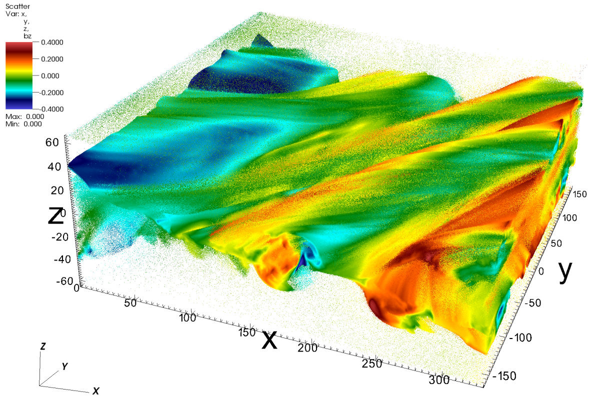
KNN: Well known applications

- Object classification in images
- Text classification and information retrieval
- Prediction of economic events
- Medical diagnosis
- 3D structure prediction of protein-protein interactions and
- Science applications

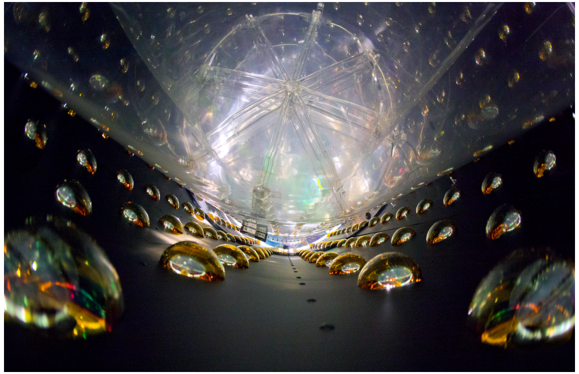
Scientific Motivation



Simulation of the Universe by the Nyx code



3D simulation of magnetic reconnection in electron-positron plasma by VPIC code

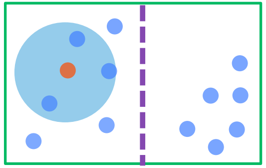


The interior of one of the cylindrical Daya Bay Antineutrino detector

Image Courtesy: Prabhat [LBNL], Oliver Rübel [LBNL], Roy Kalschmidt (LBNL)

KNN: K-Nearest Neighbors

- Problem: Given a set of multi-dimensional data points, find the k closest neighbors



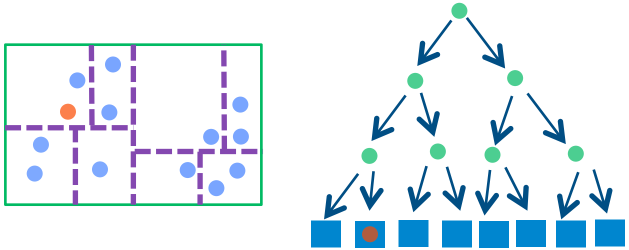
k = 3

- Naïve approach: compute distance from the query point to all points and get the k-closest neighbors
  - Pros: highly parallel
  - Cons: Lot of additional distance computation, not suitable for large scale dataset (communication heavy)

Can we take advantage of the partitioning?

KNN: kd-tree

❖ Use kd-tree, a space-partitioning data structure for organizing points in a k-dimensional space.

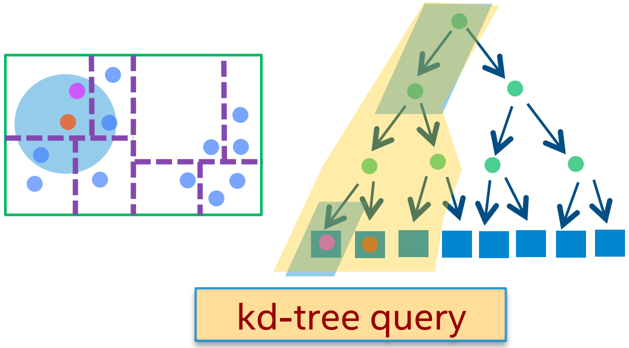


kd-tree construction

```
function kd-tree(Points* P)
    select an axis (dimension)
    compute median by axis from the P
    create a tree node
    node->median = median
    node->left = kd-tree (points in P before median)
    node->right = kd-tree (points in P after median)
```

KNN: kd-tree

❖ Use kd-tree, a space-partitioning data structure for organizing points in a k-dimensional space.




```
function find-knn(Node node, Query q, Results R)
      if node is a leaf node
            if node has a closer point than the point in R
                  add it to R
      else
            closer_node = q is in node->left ? node->left : node->right
            far_node    = q is in node->left ? node->right : node->left
            find-knn(closer_node, q, R)
            if(farthest point in R is far from median)
                   find-knn(far_node, q, R)
```

KNN

KNN has two steps:

1. kd-tree construction => build the kd-tree
2. query => to compute the k-nearest neighbors

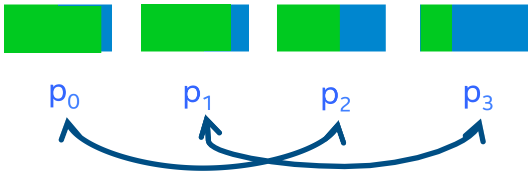
What are the challenges in parallelizing KNN!

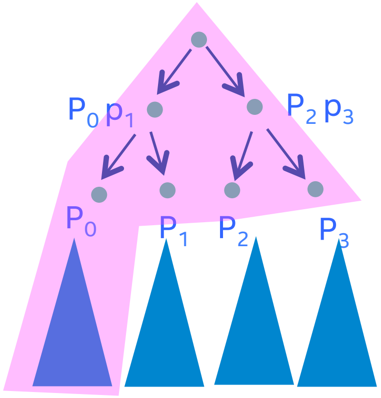
KNN: distributed kd-tree construction

- At each node:
  compute median of a selected dimension
  move points to the left node and right node

- Each internal node contains
  - Selected dimension
  - Median
  - Left pointer and right pointer

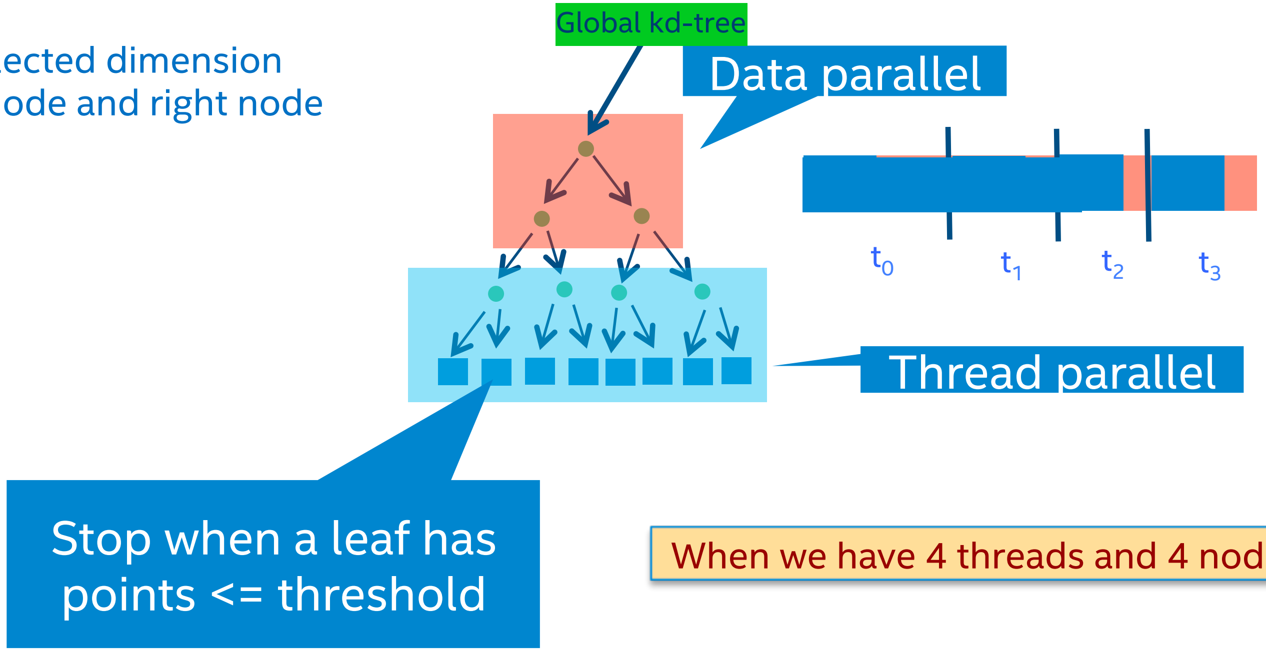
- Each leaf node
  - contains a set of points





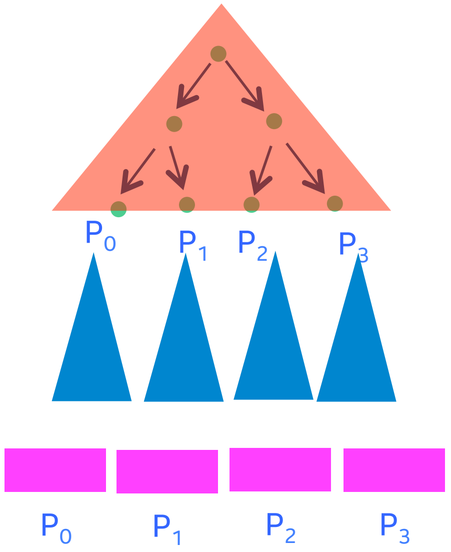
KNN: local kd-tree construction

- compute median of a selected dimension
- move points to the left node and right node

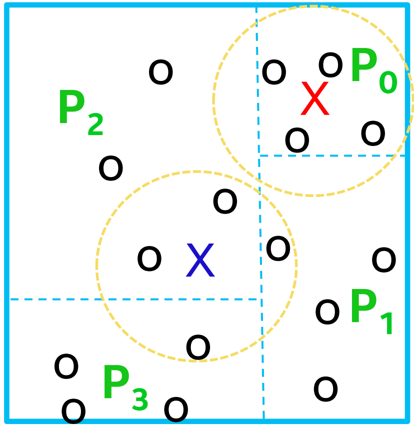


When we have 4 threads and 4 nodes?

KNN: distributed kd-tree querying

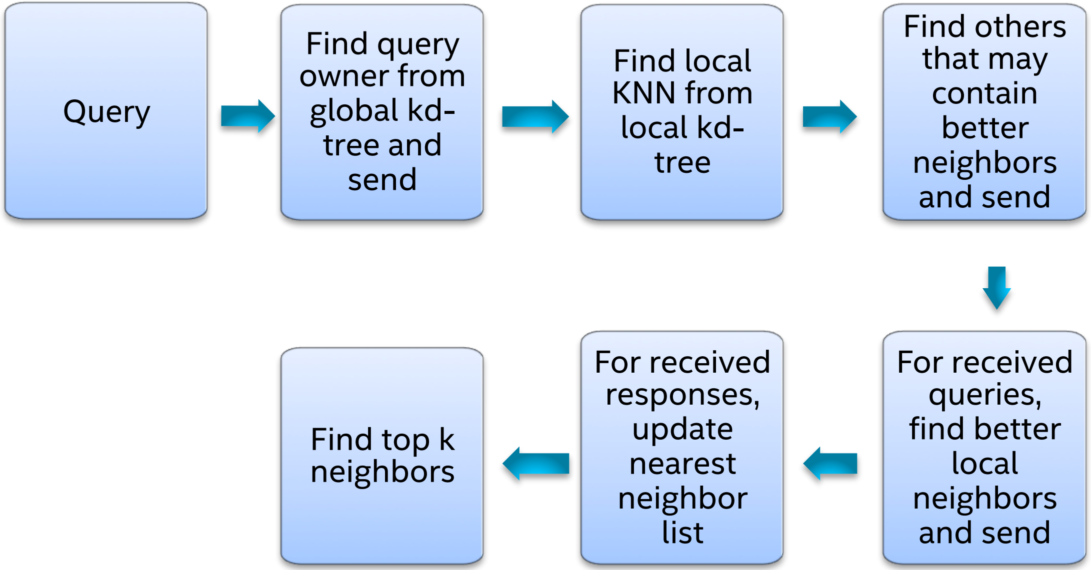


- Each processor has a set of queries
  - Queries could be local and non-local



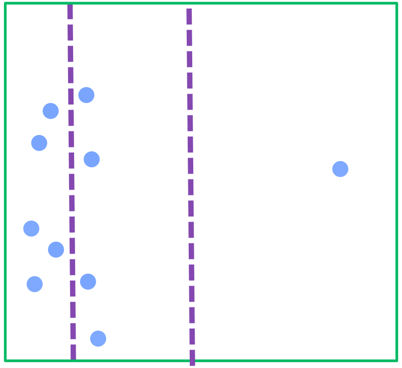
- Non-local queries:
  - Ask every node (more computation and communication)
  - Transfer ownership
    - ✓ Get the local closest k points
    - ✓ Ask neighbor node with the k^{th} point distance

KNN: distributed kd-tree querying



KNN: Algorithmic Choices [kd-tree construction]

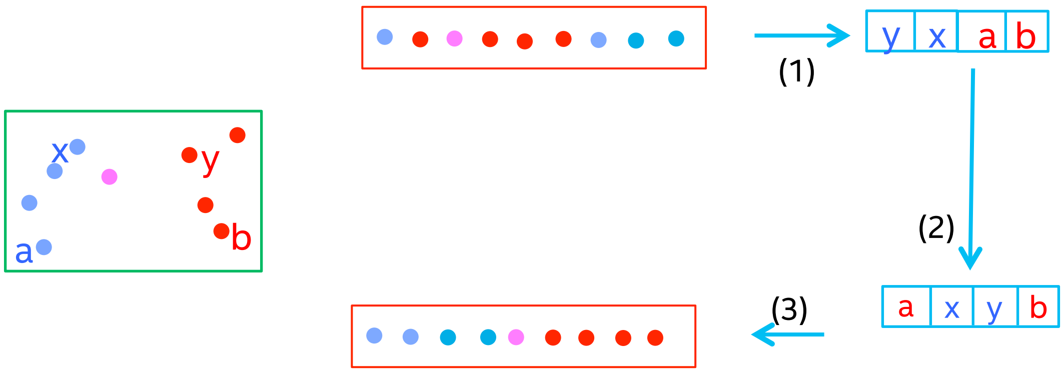
- Choice of split point
- Choice of split dimension
- Choice of bucket size



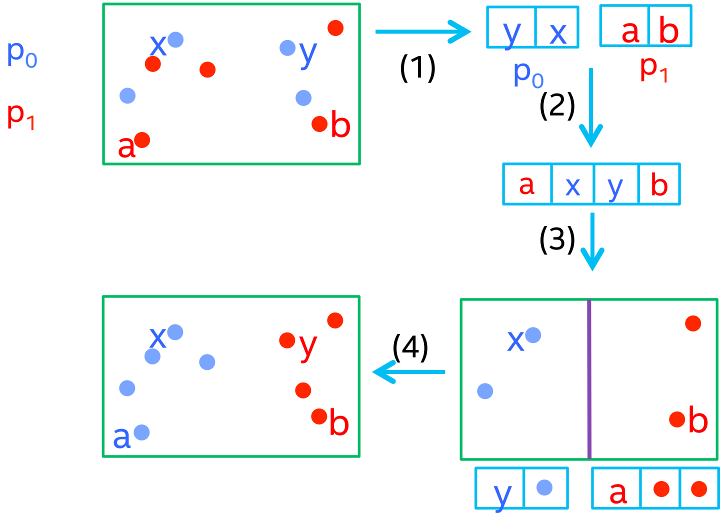
Median, average, or
min + (max-min)/2

Sampling based median computation

KNN: Sampling based median (local)

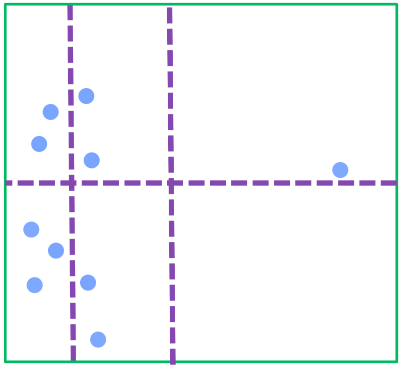


KNN: Sampling based median (distributed)



KNN: Algorithmic Choices [kd-tree construction]

- Choice of split point
- Choice of split dimension
- Choice of bucket size

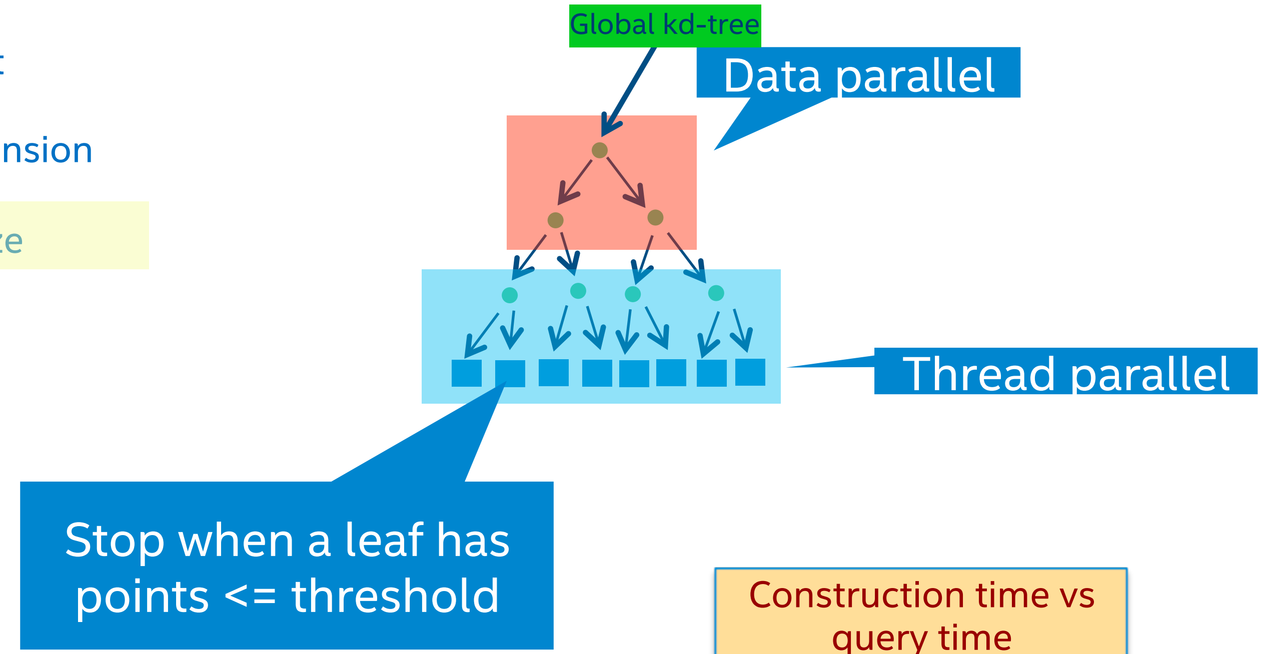


Maximum range

Imbalanced kd-tree
More communication and computation

KNN: Algorithmic Choices [kd-tree construction]

- Choice of split point
- Choice of split dimension
- Choice of bucket size



Construction time vs query time

KNN: Experiments

Datasets

Name	Particles	Dims	Time (C)	k	Queries (%)	Time (Q)	Cores
cosmo_small	1.1 B	3	23.3	5	10	12.2	96
cosmo_medium	8.1 B	3	31.4	5	10	14.7	768
cosmo_large	68.7 B	3	12.2	5	10	3.8	49152
plasma_large	188.8 B	3	47.8	5	10	11.6	49152
dayabay_large	2.7 B	10	4.0	5	0.5	6.8	6144
cosmo_thin	50 M	3	1.1	5	10	1.1	24
plasma_thin	37 M	3	1.0	5	10	0.8	24
dayabay_thin	27 M	10	1.8	5	0.5	3.2	24

- Cosmology: Three cosmological N-body simulations datasets using Gadget code [12 GB, 96 GB, 0.8 TB]
- Plasma Physics: 3D simulation of magnetic reconnection in electron position using VPIC code [2.5 TB]
- Particle Physics: Signals from cylindrical antineutrino detectors at Daya Bay experiment [30 GB]

- One representative small dataset from each to experiment on single node.

KNN: Experiments

Hardware Platform

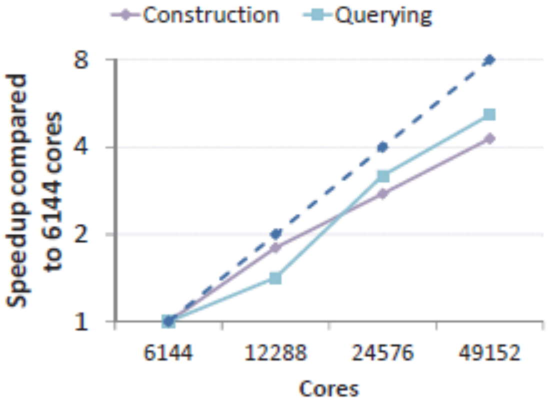
- Edison, a Cray XC30 supercomputing system @NERSC
    - 5576 compute nodes, each with two 12-cores Intel® Xeon ® E5-2695 v2 processors at 2.4 GHz and 64 GB of 1866-DDR3 memory.
    - Cray Aries interconnect (10 GB/s) bi-directional bandwidth per node

- Codes developed in C/C++
- Compiled using Intel® compiler v.15.0.1 and Intel ® MPI library v.5.0.2
- Parallelized using OpenMP (within node) and MPI (between nodes)

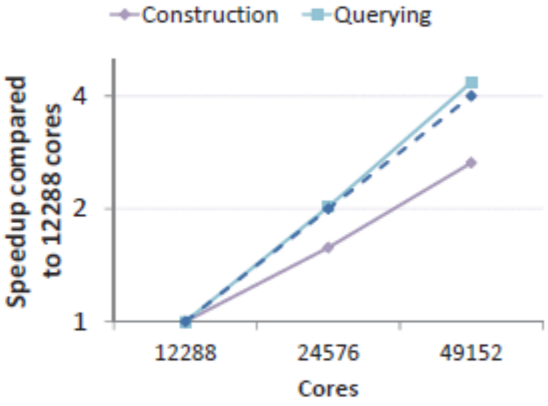
KNN: Results (Multinode)

Scalability: Strong scaling (Construction and Querying)

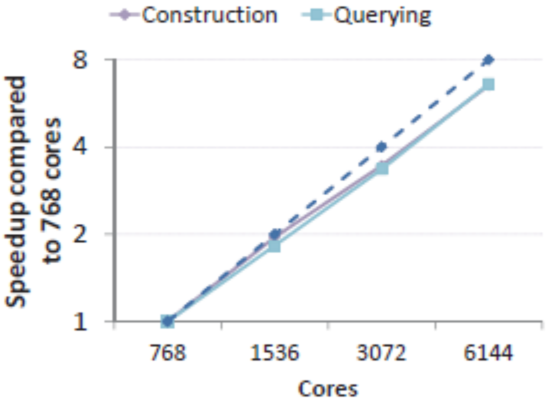
cosmo_large [69B particles], *plasma_large* [189B particles], and *dayabay_large* [3B particles]



(a) Cosmology



(b) Plasma physics



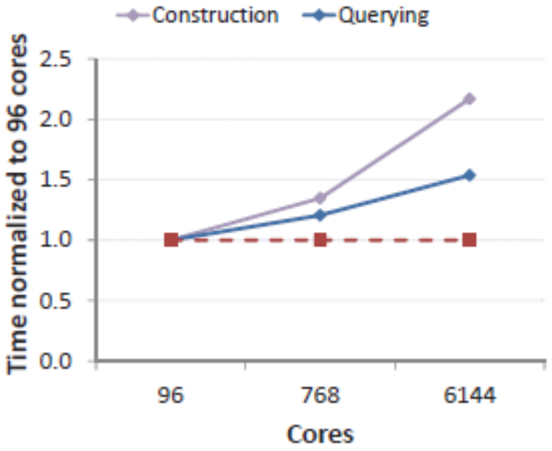
(c) Particle physics

4.3x using 8x cores 5.2x using 8x cores	2.7x using 4x cores 4.4x using 4x cores	6.5x using 8x cores 6.4x using 8x cores

KNN: Results (Multinode)

Scalability: Weak scaling (Construction and Querying)

The first three cosmology datasets

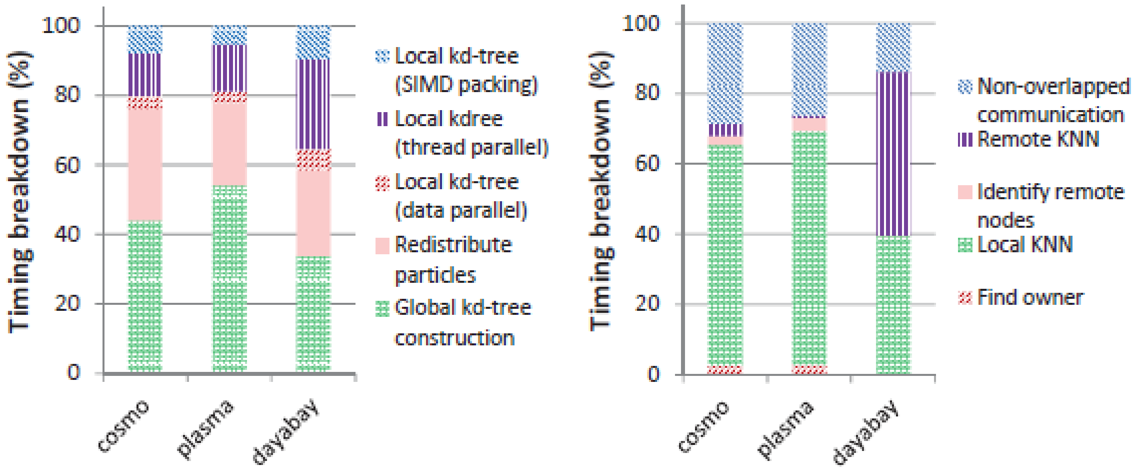


Weak scaling (Cosmology)

2.2x (1.5x) increase when scaled to 64x more cores and data

KNN: Results (Multinode)

Runtime breakdown (Construction and Querying)

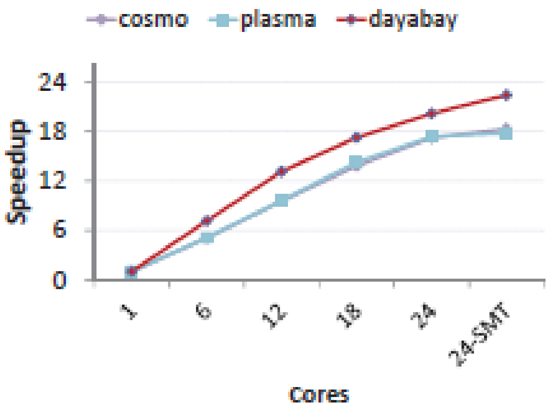


(a) Construction time (b) Querying time

cosmo_large [69B particles], *plasma_large* [189B particles], and *dayabay_large* [3B particles]

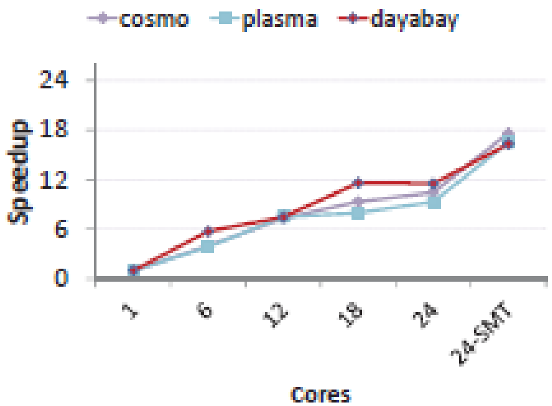
KNN: Results (single node)

Scalability (Construction and Querying)



(a) Construction

20x using 24 cores (22.4x)

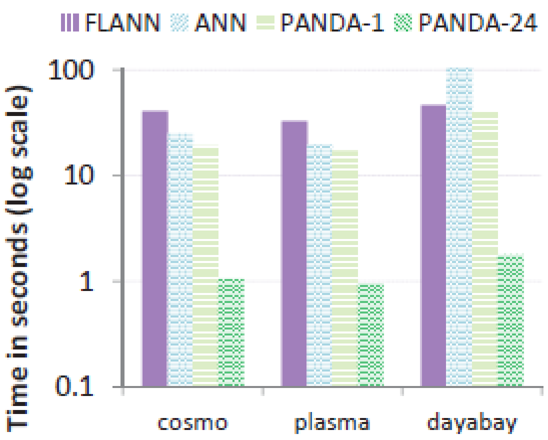


(b) Querying

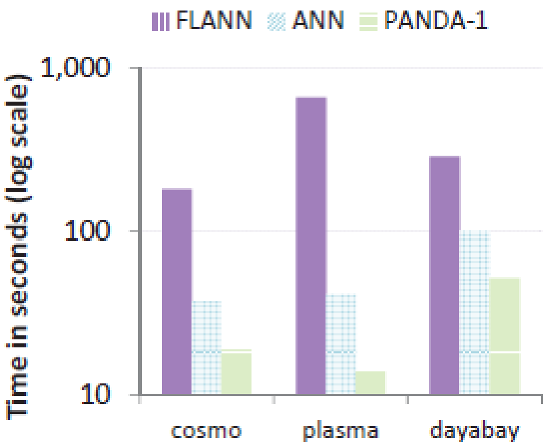
12x using 24 cores (16.2x)

KNN: Results (single node)

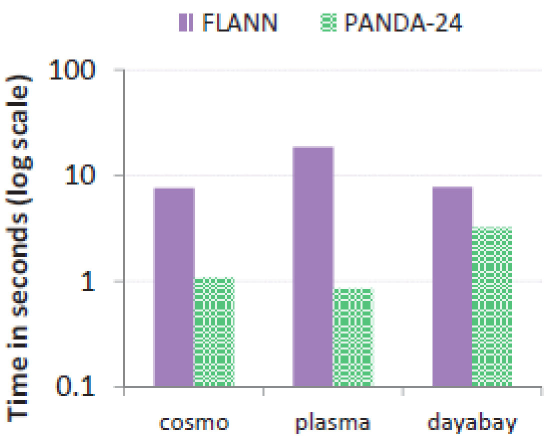
Comparison to previous implementations



(a) Training



(b) Classification (1 thread)



(c) Classification (24 thread)

2.6x on single core

59x on 24 cores

48x on single core

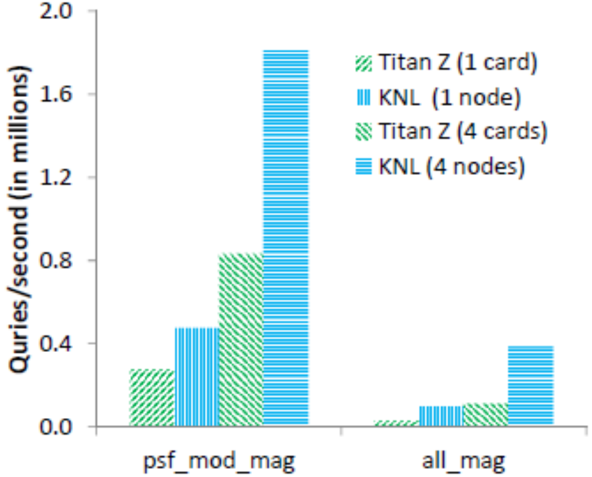
22x on 24 cores

KNN: Intel Xeon Phi (KNL) processor

Datasets used for experiments on KNL

	Construction		Querying	
Name	Particles	Dims	Particles	Dims
psf_mod_mag	2M	10	10M	10
all_mag	2M	15	10M	15
cosmo	254M	3	254M	3
plasma	250M	3	250M	3

Comparing KNL to Titan Z [1] performance



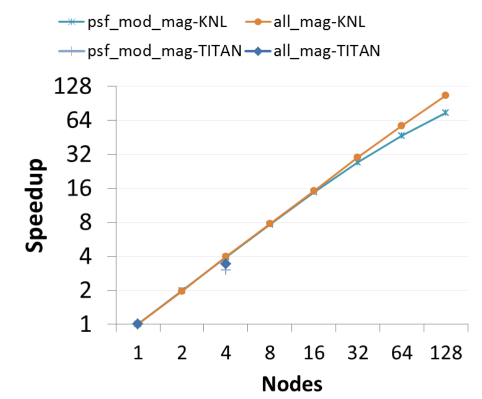
Up to 3.5X performance improvement

[1] Fabian Gieseke, Cosmin Eugen Oancea, Ashish Mahabal, Christian Igel, and Tom Heskes. Bigger Buffer k-d Trees on Multi-Many-Core Systems. http://arxiv.org/abs/1512.02831, Dec 2015

KNN: Intel Xeon Phi (KNL) processor

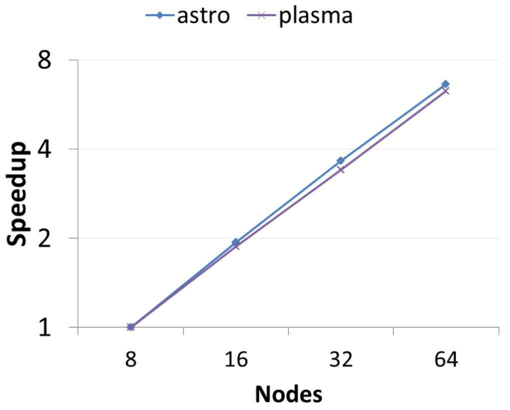
Multinode KNL

KNL Scaling



Each node keeps the entire kd-tree similar to [1]

KNL Scaling (6.5X speedup using 8X more node)



- Each node has partial view of the entire kd-tree (keeps global kd-tree and only its own local kd-tree)
- 127X larger construction dataset, 25X larger query dataset
- Titan Z [1] based implementation does not use distributed kd-tree, hence incapable to deal massive dataset

[1] Fabian Gieseke, Cosmin Eugen Oancea, Ashish Mahabal, Christian Igel, and Tom Heskes. Bigger Buffer k-d Trees on Multi-Many-Core Systems. http://arxiv.org/abs/1512.02831, Dec 2015

Conclusions

- This is the first distributed kd-tree based KNN code that is demonstrated to scale up to ~50,000 cores.
- This is the first KNN algorithm that has been run on massive datasets (100B+ points) from diverse scientific disciplines.
- We show that our implementation can construct kd-tree of 189 billion particles in 48 seconds on utilizing ~50,000 cores. We also demonstrate computation of KNN of 19 billion queries in 12 seconds.
- We successfully demonstrate both strong and weak scalability of KNN implementation.
- We showcase almost linear scalability un to 128 KNL nodes
- Our implementation is more than an order of magnitude faster than state-of-the-art KNN implementation.

Acknowledgements

- Daya Bay Collaboration for providing access to datasets
- Zarija Lukic for providing access to Gadget datasets
- Vadim Roytershteyn for providing access to VPIC datasets
- Tina Declerck and Lisa Gerhardt for facilitating large scale runs on NERSC

Thank you for your time

Prabhat

prabhat@lbl.gov

www.intel.com/hpcdevcon