kANN on the GPU with Shifted Sorting

Lance Simons
lcsimons@ucdavis.edu

Shengren Li
Jagadeesh Bhaskar Pakaravoor
Fatemeh Abbasinejad
John D. Owens
Nina Amenta

The Elevator Pitch

We answer batched kANN queries using *shifted sorting*, from Liao, Lopez, and Leutenegger.

The algorithm is well-suited to the GPU, so our implementation is in CUDA.

We compare our speed and accuracy against FLANN, which can also use CUDA.

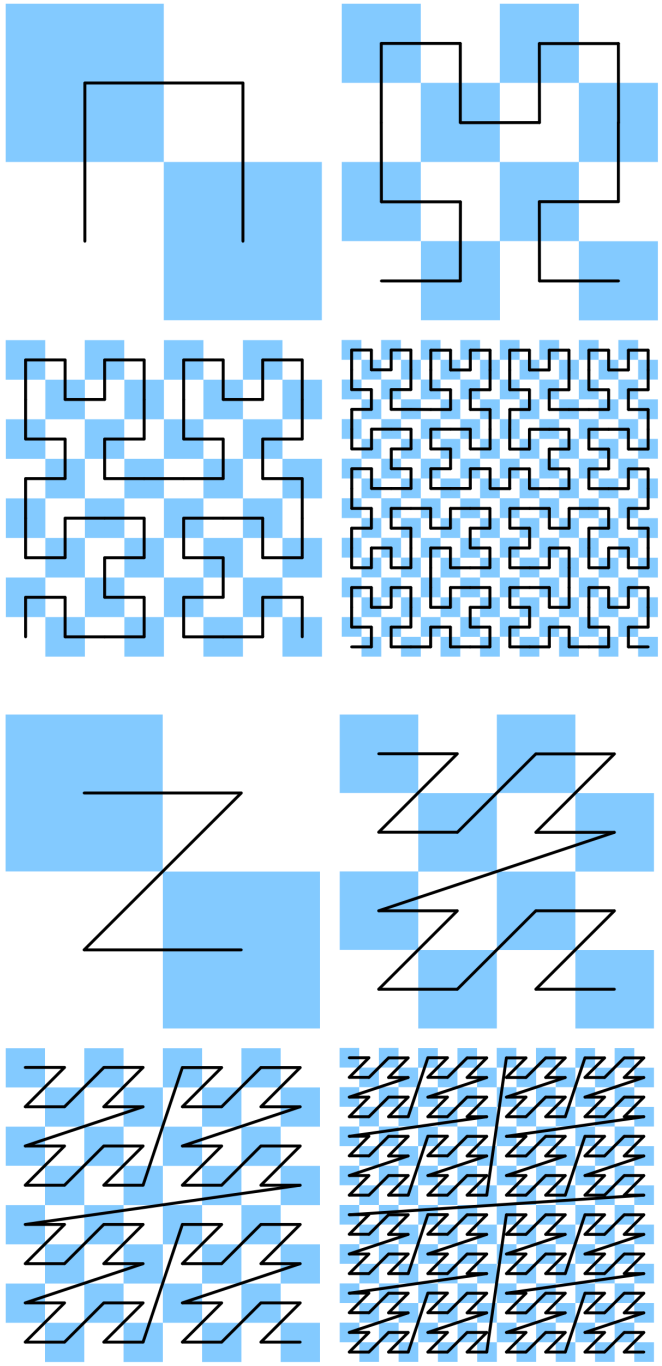
Background
Hilbert and Morton

Represent an octree decomposition of the space.

Specify traversal order.

Hilbert exhibits better locality.

Morton is easier to calculate.



The *Shifted Sorting* Algorithm

The data points are shifted and stored in lists in Hilbert Code order.

In 3D:

Each shift is in the (1,1,1) direction

5 shifts are required

Each query searches into the lists, selecting the k data points before and after the query's Hilbert Code.

From the 10k results, return the best k.

How we differ from Liao et al.

We use Morton Codes instead of Hilbert Codes.

Hilbert Codes are more accurate.

Hilbert Codes are harder to calculate.

- MCs are a simple interleaving of bits:
  - Can easily be done in both directions
  - Allows fast approximation of the x,y,z coords
  - Easier to generalize to higher dimensions

How we differ from Liao et al.

We prevent binary searches by sorting query and data MCs together.

The least-significant bit of the MC indicates query/data, making scan/compact easy.

We can then compact the data MC list, which each query reads the $2k$ candidates from.

How we differ from Liao et al.

We only store a single sorted list of Morton Codes, instead of five. This requires maintaining a list of the k best candidates we've seen so far.

At each iteration:

- Shift data and queries, sort by Morton Code
- Find 2k candidates for each query
- Merge the 2k candidates into the current k

Why target the GPU?

The vast majority of the algorithm is data-independent. The lack of divergent control flow means the data-parallel nature of CUDA works very well here.

Merging is the only step that is data-dependent, due to collisions during atomic operations.

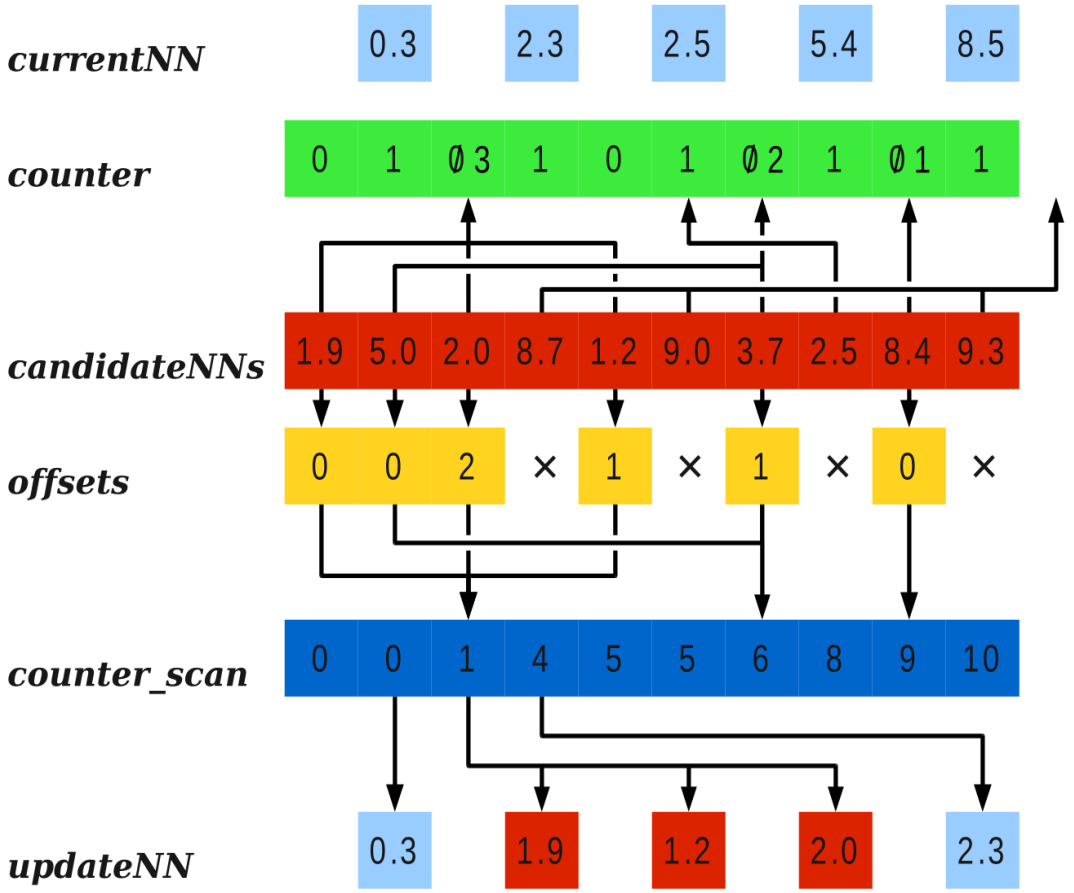
The Merge Step

Initialize Counter
- Even → 1
- Odd → 0

Binary Search
- Candidates search into *currentNN*

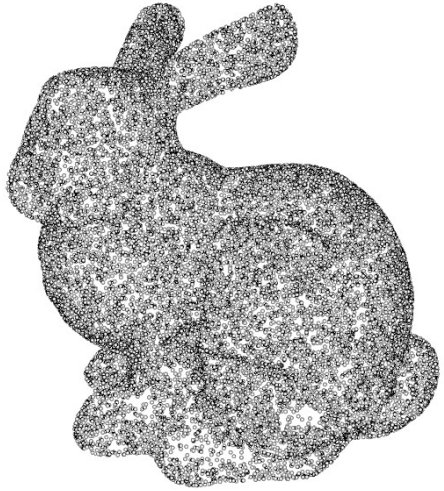
Exclusive Scan

Final Position
- Scan + Offset

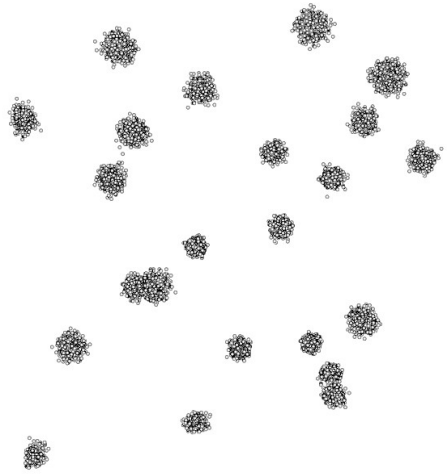


Test Datasets

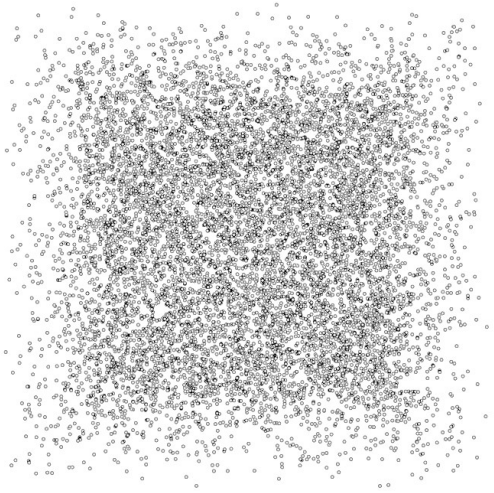
Bunny



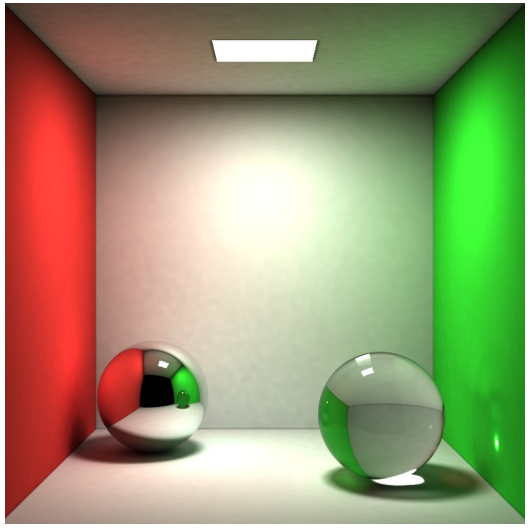
Cluster



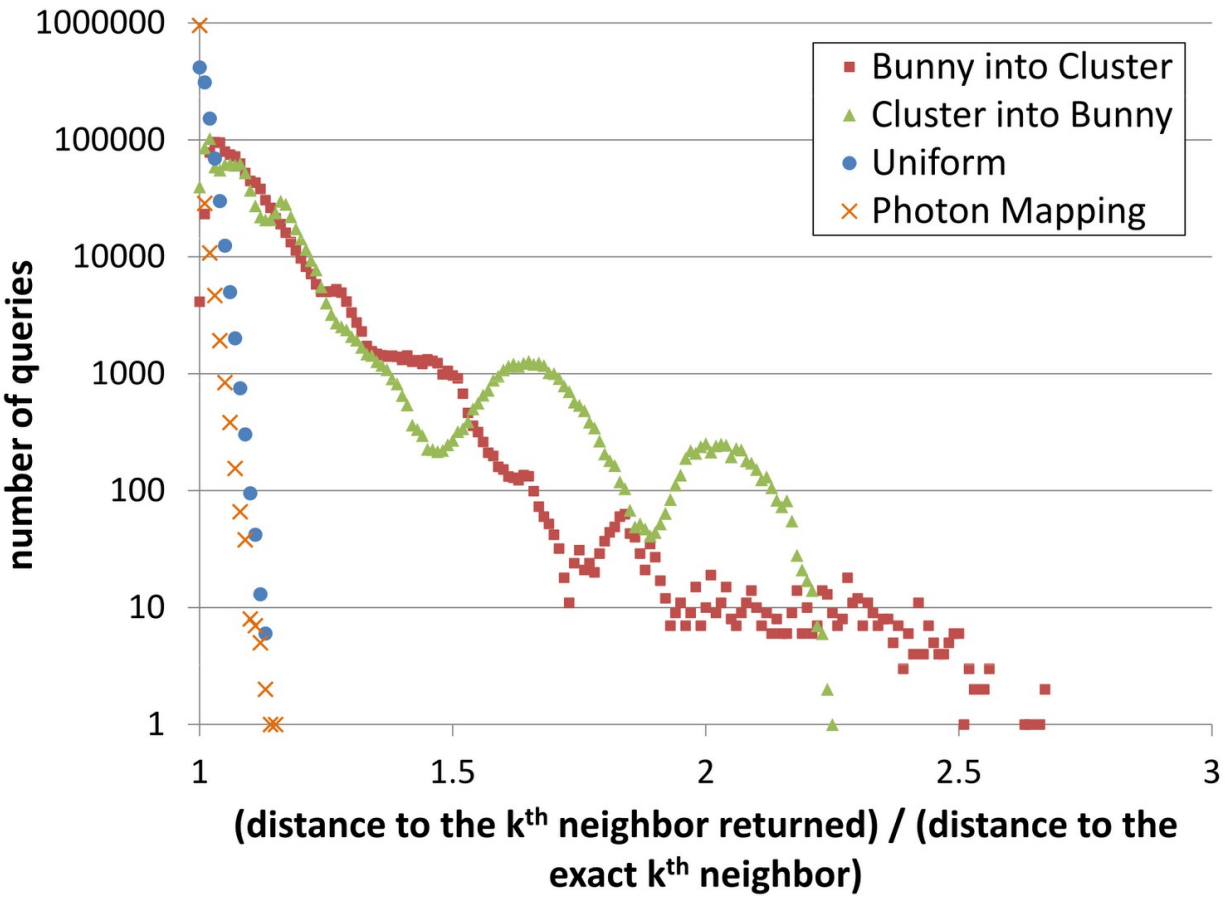
Uniform



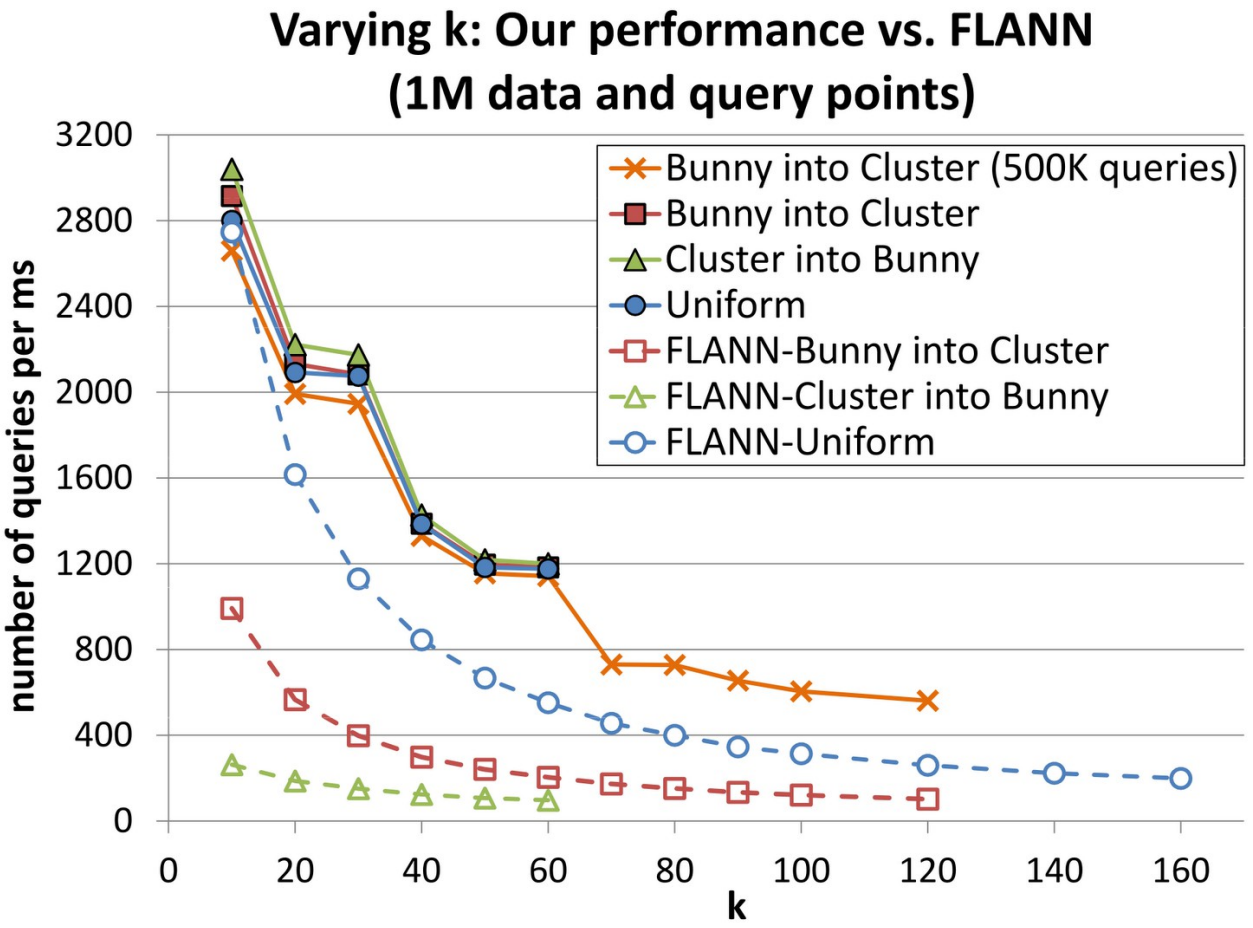
Photon Mapping



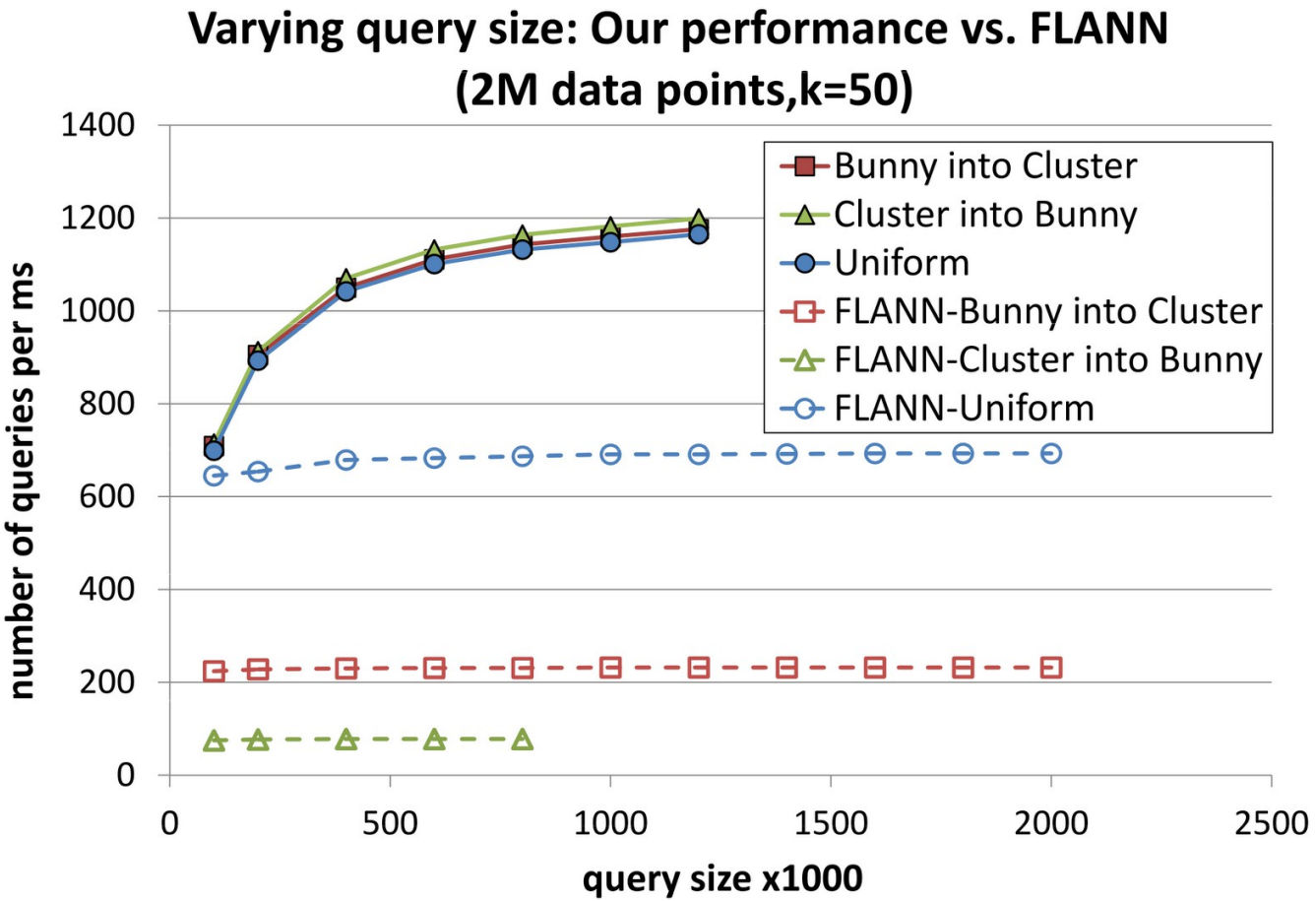
Results - Accuracy



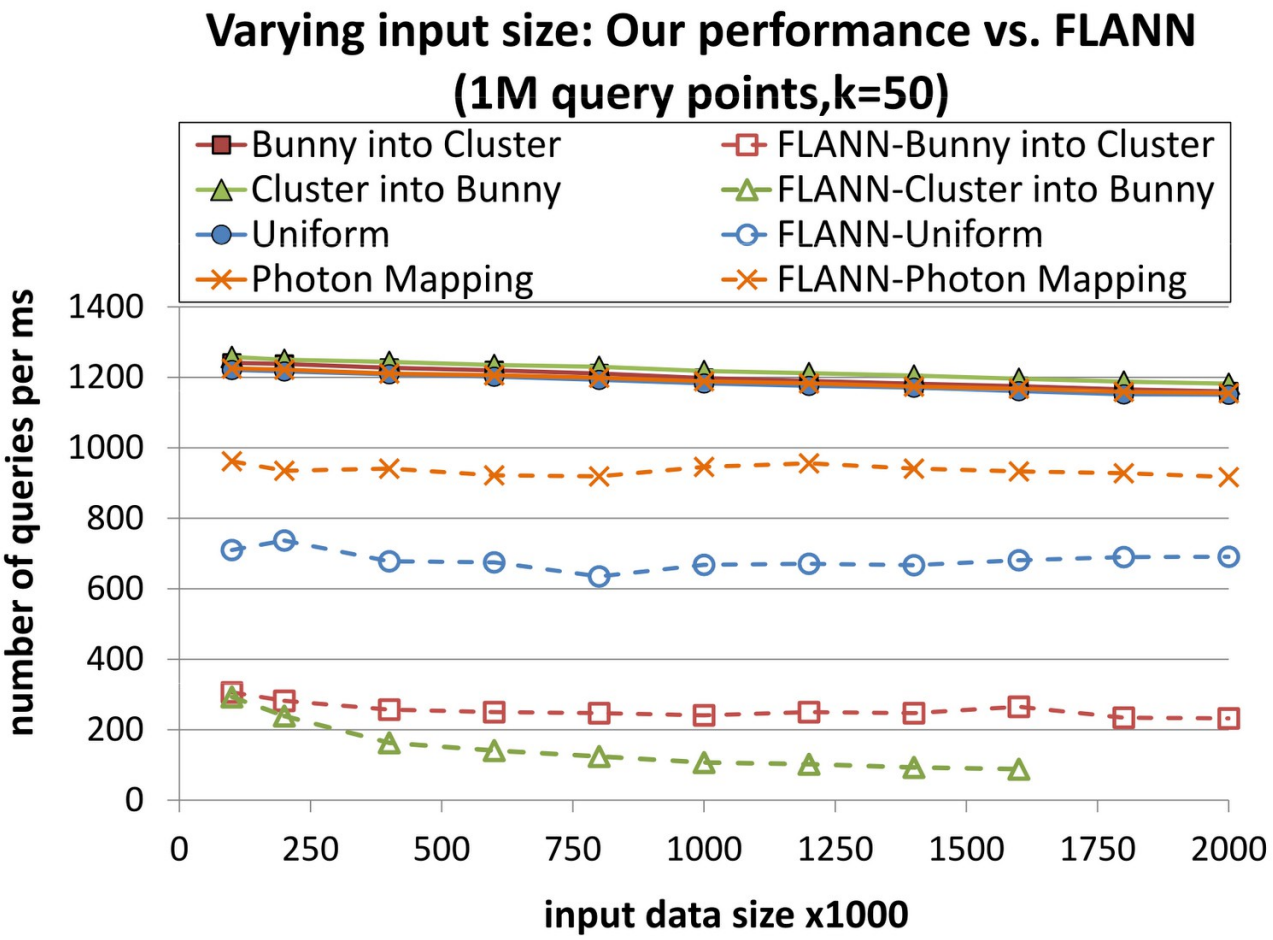
Results - Timings



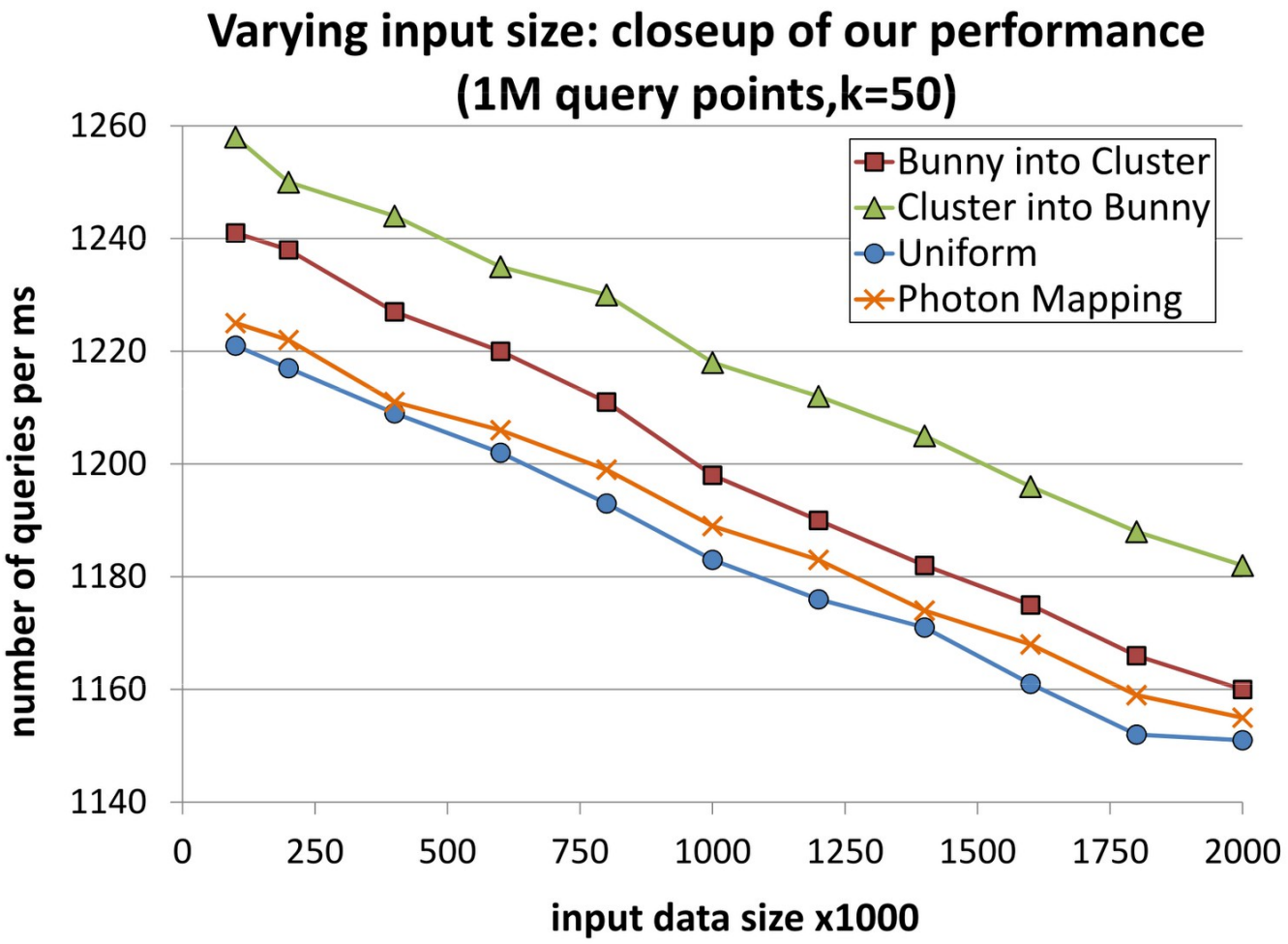
Results - Timings



Results - Timings



Results - Timings



Future Directions

Higher Dimensions

Aggregation Kernel

Kernel Fusion / Shared Memory

- Less static memory usage
- Possible to return results as computed, instead of *en masse*.