A CUDA Based Parallel Decoding Algorithm for the Leech Lattice Locality Sensitive Hash Family

Lee Carraher

Department of Computer Science
University of Cincinnati

May 8, 2012
Outline

- Thesis Goal
- Nearest Neighbors Problem and Applications
- LSH
- Parallel Leech Decoder
- Results
- Conclusions and Future Directions
Our Goal

The goal of this thesis is to present a CUDA optimized parallel implementation of a bounded distance Leech Lattice decoder \(^1\) for use in query optimized c-approximate k-nearest neighbors using the locality sensitive hash framework of E2LSH \(^2\).

In addition, we will present an improvement to the E2LSH Algorithm that improves the performance for real world data problems by an order of magnitude in regards to selectivity, with no change in algorithmic or storage complexity.

---


The Nearest Neighbor Problem

Definition (Exact NN)

Given a set of points $P$ in $\mathbb{R}^d$ and query point $q$ return a point $p \in P$ such that $p = \text{Argmin} \{\rho(p, q)\}$, where $\rho$ is some metric function.\(^3\)

And an extension to an arbitrary number of neighbors.

Definition (k-Nearest Neighbor Search)

Given a set of points $P$ in $\mathbb{R}^d$ and query point $q$ return the $k$ nearest points $p \in P$ to $q$.\(^4\)

---

\(^3\) H. Samet, *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006

Applications of NN

- Pattern Recognition
- Image/Object Recognition - Robot Vision systems, object and motion tracking
- Genomics - similar and exact gene sequences in long DNA reads
- Biology/Medical - drug interaction and symptoms
- Recommender Systems - similar items based on user submission
- Data Compression
Definition (Linear Search NN)

Given a set of points $P$ in $\mathbb{R}^d$ and query point $q$ return the nearest points $p \in P$ to $q$ by minimizing the function $f() = \rho(P, q)$, where $\rho$ is some distance function.

This algorithm has complexity $\Theta(nd)$. And for large $d$ and $n$, becomes computationally expensive. For this reason, it is not often used in practice and instead kd-tree based NN search is used.
Kd Trees for NN search

Kd-trees are an extension of a binary tree in which nodes correspond to points in the database and splitting decisions are based on the value of the point along one of its axis’s. This method forms splitting planes which partition the points in $\mathbb{R}^d$. A simple 2d example is given below.

```
[3,5]  [2,2]  [5,5]
[2,1]  [1,2]  [4,6]
```
The search algorithm consists of recursively searching the binary tree.

In 1 dimension, this is equivalent to a binary search, and is optimal.

However, as $d \to \infty$, the recursion approaches linear search.
The Curse of Dimensionality

Although there are many interpretations of the common **COD** in regards to nearest neighbor searching it applies to an exact search method’s complexity approaching linear search complexity as d grows large. This is due to the chosen splitting plane having an increasingly lower probability of partitioning the data equally.

Consider the distance between any two points \(x\) and \(y\). Under Euclidean distance, \(dist(x, y) = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2}\) Now if we consider the vector as being composed of values that have a uniformly distribution, according to Ullman,\(^5\) the range of distances are all very near the average distance between points. Given this, it becomes very difficult to differentiate between near and far points.

---

COD is sometimes cited as the cause for the distance function loosing its usefulness for high dimensions.\textsuperscript{6} This arises from the ratio of metric space partitioning to hypersphere embedding.

\[
\lim_{d \to \infty} \frac{\text{Vol}(S_d)}{\text{Vol}(C_d)} = \frac{\pi^{d/2}}{d2^{d-1} \Gamma(d/2)} \to 0
\]

Given a single distribution, the minimum and the maximum distances become indiscernible.\textsuperscript{7} Or the relative majority of space is outside of the sphere.

\textsuperscript{6} K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, “When is ”nearest neighbor” meaningful?,” in \textit{In Int. Conf. on Database Theory}, pp. 217–235, 1999

\textsuperscript{7} K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, “When is ”nearest neighbor” meaningful?,” in \textit{In Int. Conf. on Database Theory}, pp. 217–235, 1999
Approximate NN Methods

Due to COD’s effects, approximate methods are often used to not only solve the problems of increasing computational complexity, but they also allow formulations of the nearest neighbor problem that somewhat side step the space embedding issue by casting them as a probabilistic problem. Many variants of Kd-Trees exist, and generally apply to approximate linear search methods.

Definition (c-Approximate Nearest Neighbor Search)

Given a set of points $P$ in $\mathbb{R}^d$ and query point $q$ return a point $p \in P$ such that its distance to $q$ is less than $c\rho(q,p)$ where $p$ is the exact solution, and $\rho$ is some distance metric.\(^8\)

*For Kd-Trees, common heuristic algorithms are best-bin first, and randomized Kd-Tree.*

\(^8\) H. Samet, *Foundations of Multidimensional and Metric Data Structures.* Morgan Kaufmann, 2006
Although approximate KD-Tree searches offer various solutions to the growing search complexity issue, they either lack guarantees of exact search complexity bounds or fixed approximation values. For this reason, we consider the LSH search method of \(^9\).

Andoni and Indyk state a query and space complexity of \(O(n)\)-space and \(O(n^\rho)\)-time where \(\rho\) is parameter of the hash function that is bounded by \(\frac{1}{4}\).

The general concept of LSH based NN lies in the definition of a locality sensitive hash function.
LSH Hash Families

**Definition (Locality Sensitive Hash Function)**

Let $\mathbb{H} = \{h : S \rightarrow U\}$ is $(r_1, r_2, p_1, p_2)$-sensitive if for any $u, v \in S$

1. if $d(u, v) \leq r_1$ then $Pr_{\mathbb{H}}[h(u) = h(v)] \geq p_1$
2. if $d(u, v) > r_2$ then $Pr_{\mathbb{H}}[h(u) = h(v)] \leq p_2$

For this family $\rho = \frac{\log p_1}{\log p_2}$
Using the above family of hash functions, the LSH algorithm can be defined as an iterative convergence of probabilities leading to a solution to the approximate near neighbor problem.

Require: $X = \{x_1, ..., x_m\}$, $x_k \in \mathbb{R}^n$ a set of points

- $\mathbb{H}$ is a locality sensitive hash function
- $D$ - is a set of buckets

for all $x_k \in X$ do

  Add $x_k$ to the hash bucket $\mathbb{H}(x)$

end for

return $\mathbb{H}, D$
Require: $H, D, \hat{x} \in \mathbb{R}^n$ a set of points
return $D[H(\hat{x})]$

If $H$ is defined as splitting the vectors over a particular random plane, the results are very similar to the approximate kd-tree algorithm with depth 1.

To improve this algorithm, we must come up with a better locality sensitive hash functions than simply splitting the dataset.

- select multiple planes and then intersect the returned hash buckets
- to increase selectivity of the buckets, we could concatenate hashes to create longer keys and more specific buckets
An Example Hash Family

Figure: Random Projection of $\mathbb{R}^2 \rightarrow \mathbb{R}^1$
Voronoi Tiling

An optimal partitioning in 2-dimensions is the Voronoi partitioning, and with point location search can be done with complexity $\Theta(\log(n))$ and linear space$^{10}$. 

Figure: Voronoi Partitioning of $\mathbb{R}^2$

Voronoi diagrams make for very efficient hash functions in 2d because, by definition, a point within a Voronoi region is nearest to the regions representative point.

Voronoi regions provide an optimal solution to the NN partitioning in 2-d Space!

However, for arbitrary dimension d, Voronoi diagrams require $\Theta(n^{d/2})$-space, and no known optimal point location algorithms exists.
Lattices

Instead we will consider lattices, which provide regular space partitioning and scale to arbitrarily large dimensional space, and have sub-linear nearest center search algorithms associated with them.

**Definition (Lattice in $\mathbb{R}^n$)**

let $v_1, ..., v_n$ be $n$ linear independent vectors where $v_i = v_{i,1}, v_{i,2}, ..., v_{i,n}$ The lattice $\Lambda$ with basis $\{v_1, ..., v_n\}$ is the set of all integer combinations of $v_1, ..., v_n$ the integer combinations of the basis vectors are the points of the lattice.

$$\Lambda = \{z_1v_1 + z_2v_2 + ... + z_nv_n | z_i \in \mathbb{Z}, 1 \leq i \leq n\}$$

---

Examples in 2D

Figure: Square(left) and Hexagonal(right) Lattices in $\mathbb{R}^2$
Finding The Nearest Representative in Constant Time

➤ Certain lattices allow us to find the nearest representative point in constant time.
➤ For example the above square lattice.
  ➤ The nearest point can be found by simply rounding our real valued point to its nearest integer.
➤ With the exception of a few exceptional lattices, more complex lattices have more complex searches (exponential as $d$ increases $^{12}$).

Exceptional Higher Dimensional Lattices

The previous lattices work well in $\mathbb{R}^2$, but our data spaces are in general $\gg 2$.

- Fortunately there are some higher dimensional lattices, with efficient nearest center search algorithms.
- $E_8$ or Gosset’s Lattice, is one such lattice in
- It is also the densest lattice packing in $\mathbb{R}^{813}$. 
Example of decoding $E_8$

$E_8$ can be formed by gluing two $D_8$ integer lattices together and shifting by a vector of $\frac{1}{2}$. This gluing of less dense lattices and shifting by a “glue vector” is a common theme in finding dense lattices.

- Decoding $D_8$ is simple
- $E_8 = D_8 \cap D_8 + \frac{1}{2}$
- both cosets of $D_8$ can be computed in parallel
- $D_8$’s decoding algorithm consists of rounding all values to their nearest integer value s.t they sum to an even number
Example of decoding $E_8$

define $f(x)$ and $g(x)$ to round the components of $x$, except in $g(x)$ we round the furthest value from an integer in the wrong direction.

Let

$$x = <0.1, 0.1, 0.8, 1.3, 2.2, -0.6, -0.7, 0.9>$$

then

$$f(x) = <0, 0, 1, 1, 2, -1, -1, 1>, \text{sum} = 3$$

and

$$g(x) = <0, 0, 1, 1, 2, 0, -1, 1>, \text{sum} = 4$$

since $g(x)$ is even, it is the nearest lattice point in $D_8$
Example of decoding $E_8$ conti.

- However we want the nearest in $E_8$, so we also have to include the coset $\cap D_8 + \frac{1}{2}$.
- We can do this by subtracting $\frac{1}{2}$ from all the values of $x$.

\[
\begin{align*}
f(x - \frac{1}{2}) &= \langle 0, 0, 0, 1, 2, -1, -1, 0 \rangle, \text{ sum } = 1 \\
g(x - \frac{1}{2}) &= \langle -1, 0, 0, 1, 2, -1, -1, 0 \rangle \text{ sum } = 0
\end{align*}
\]

Now we find the coset representative that is closest to $x$ using a simple distance metric.

\[
\|x - g(x)\|^2 = 0.65
\]

\[
\|x - g(x - \frac{1}{2})\|^2 = 0.95
\]

So this case it is the first coset representative:

\[
\langle 0, 0, 1, 1, 2, 0, -1, 1 \rangle
\]
Leech Lattice

By gluing sets of E8 together in a way originally conceived by Curtis’ MOG, we can get an even higher dimensional dense lattice called the Leech lattice.
Here we will state some attributes of the leech lattice as well as give a comparison to other lattices by way of $Eb/N_0$ and the computational cost of decoding.

Some Important Attributes:

- Densest Regular Lattice Packing in $\mathbb{R}^{24}$
- Lattice Construction can be based on 2 cosets $G_{24}$
- Sphere Packing Density: $\frac{\pi^{12}}{12!} \approx 0.00192957$
- $K_{min} = 196560$
Figure: Performance of Some Coded and Unencoded Data Transmission Schemes
Leech Lattice Decoding

Some information about the decoding algorithm:

- The decoding of the leech lattice is based closely on the Decoding of the Golay Code.
- In general, advances in either Leech decoding or binary Golay decoding imply an advance in the other.
- The decoding method used in this implementation is based on Amrani and Be’ery’s ’96\textsuperscript{14} publication for decoding the Leech lattice, and consists of around 519 floating point operations and suffers a gain loss of only 0.2bB.
- In general decoding complexity scales exponentially with dimension.\textsuperscript{15}

Next is an outline of the decoding process.


Leech Decoder

Figure: Leech Lattice Decoder
CUDA is a programming/computational platform for performing general purpose processing on GPU hardware. CUDA is implemented as an extension of C, exposing various parallelization and memory allocation primitives. The advantages of GPGPU computing can be most easily seen in a picture on the next slide.
Advantages of CUDA in a Picture

Figure: Transistor Allocation for CPU vs. GPU Architectures
CUDA doesn’t come without a cost

So what do we have to pay for all of these processors and overall higher ALU density?

▶ We have to adapt or rewrite programs into CUDA
▶ Memory Latency, no caching* (feeding lots of little beasts)
▶ Memory size per core
▶ Synchronization, communication, standard parallel programming costs
Parallel Decoder Outline

- Naive CUDA Implementation
- Improved CUDA Implementation
- CUDA Pitfalls
- Some Performance Tweaks
Preprocessing

**Require:** $X = \{x_1, ..., x_m\}$, $x_k \in \mathbb{R}^n$

1: $U(x)$ is a universal hash function
2: $h_k(x) \in \mathbb{H}$ is $(r, cr, p_1, p_2)$-sensitive
3: choose $l$ s.t.
4: $G \leftarrow i \in \mathbb{Z}^l$ from $[0, n)$
5: $g(x) = \{h_1(x), ..., h_j(x)\}$
6: $D \leftarrow []$
7: **for all** $x_k \in X$ **do**
8: $D \leftarrow U(g(x_k))$
9: **end for**
10: **return** $G, D$
Query: c-approx k-nearest neighbor

\textbf{Require: } \hat{x} \in \mathbb{R}^n, G, H, U(), D

1: \textbf{for all } g_j(x) \in G \textbf{ do}
2: \quad U(g_j(\hat{x}))
3: \textbf{end for}
4: K = []
5: \textbf{for all } l \in L \textbf{ do}
6: \quad d \leftarrow \text{dist}(\hat{x}, D[l])
7: \quad K \leftarrow \{d, D[l]\}
8: \textbf{end for}
9: \text{sort}(K)
10: \textbf{return } K[0 : k]
An addition to the Standard Leech Lattice LSH Algorithm

- As suggested in Panigrahy\textsuperscript{16}, minimize database size (to $\Theta(n)$) by delaying replication to the query phase.
- In addition we use a suggestion of Jegou\textsuperscript{17} for E8 exact $c$-nearest neighbor search, by ordering our 2L exact search list by the lattice center weights returned by the leech lattice algorithm.


\textsuperscript{17}
Scale invariant feature transform is an image transform patented by David Lowe of Carnegie Mellon University, for identifying local features in an image that are invariant to scale, rotational, affine and to some extent lighting transformations.

SIFT maps an image to a set of feature vectors, by first finding keypoints and then applying various rotational and scale transformations such that the now feature point is scale and rotation invariant.

These features will serve as our database vectors, and a query image will be SIFT’d and its features will be searched in the database using the Lattice LSH search Algorithm.
SIFT Example

Figure: Top image is a scene and the bottom contains the SIFT image matches
Results

Test data was performed on real world SIFT vectors formed from the Caltech256\textsuperscript{18} Database. The images in the imageset were converted to their SIFT vector equivalents and an LSH database was created from the SIFT Vectors relating images. Random images were chosen and searched with our parallel LSH Algorithm, as well as an exhaustive search for nearest neighbors based on the default algorithm provided with the SIFT library.

Query Time as a Ratio of Parallel to Total Time

Average speedup in decoding between parallel and sequential decoding is approximately 41x.

Figure: Ratio of Parallel to Total Time for Queries on an LSH Search Database using Real World SIFT Sample Data
Extrapolated Lowes Linear vs Parallel LSH Search

Below is a semilog comparison of linear vs. parallel LSH Search. Extrapolation is required due to the rapidly increasing complexity of linear search.

Figure: Extrapolated Average Search Query Times
Extrapolated Lowes Linear vs Parallel LSH Search

Figure: Recall Accuracy of Random SIFT Vector Image Query
Effects of our Addition 2 and Selectivity

![Graph: Data Recall Selectivity for Query Adaptive and Unranked LSH](image)

**Figure:** Data Recall Selectivity for Query Adaptive and Unranked LSH
Conclusions

- Our primary conclusion is that parallel decoding is a useful method for decoding the leech lattice, however its utility is not scalable to arbitrarily large databases in this implementation due to the sequential bottleneck.

- On a more positive note, the query adaptive implementation yields a considerable benefit to selectivity over the original method.
Future Directions

Although the applications of approximate nearest neighbor search are manifold, offering a wide variety of similar applications above, we will instead focus on directions that better accommodate this systems strengths and avoiding its weaknesses.

▶ Adaptive Mean Shift
▶ Networked Distributed KNN Search for Heterogeneous Large Scale Problems
▶ Finding more rigorous correlations between ECCs and Locality Sensitive Hashing Functions in regards to $\Theta(n^p)$. 
Without explicitly defining the functions involved in Mean Shift and the variant Adaptive Mean Shift, we will simply show the update gradient step below. and define $N(x)$ as the set of nearest neighbors to $x$.

\[
\vec{x} = \frac{\sum_{x_i \in N(x)} K(x - x_i) \vec{x}}{\sum_{x_i \in N(x)} K(x - x_i)}
\]

A simple extension would be to use LSH-KNN at this step, furthermore the approximation factor of $c$, may even be somewhat beneficial in real world data as it may add a beneficial soft margin to the clustering.
Distributed KNN ...

- This extension may very well be used as a jumping point into parallel exa-scale computing for my further research.
- A direction I would like to pursue is to use the universally hashed, locality sensitive hash values for nearest neighbor vectors, and fold them into a search across heterogeneous systems using the same hashing scheme and supporting a query architecture similar to internet routing schemes such as RIP or the more scalable OSPF (open shortest path first) algorithm.
- As an added bonus, there seems to be a variety of tunable attributes
  - space-time tradeoff
  - actual network architecture versus algorithmic architecture and optimizations.
Closing Notes

Thank you everyone for your time! Complete References as well as slides, source code, and original thesis are freely available at homepages.uc.edu/carrahle


