Parallel Search of k-Nearest Neighbors with Synchronous Operations

Nikos Sismanis[1] Nikos Pitsianis[1,2] Xiaobai Sun[2]

[1]Dept. ECE
Aristotle University, Greece

[2]Dept. CS
Duke University, USA

September 11, 2012

Outline

KNN search: Primitive and Prevalent Operation

Identification of most matching points from a large and high dimensional data space/corpus, according to a well defined distance measure

More applications with increased data acquisition for

- machine learning and modeling
- pattern matching and (speech, image) recognition
- filtering or localization in data analysis & mining

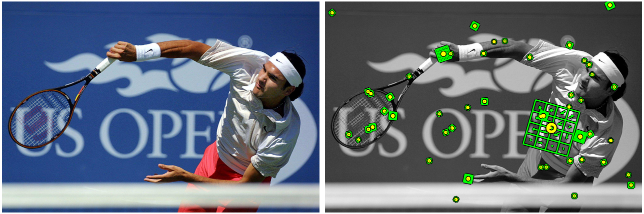
Facilitating various research areas: computer/machine vision, computer-human interaction, computational imaging, geometry, computational statistics

KNN Search for Image Queries

[1] D. G. Lowe, Inter. J. Comp. Vis., 2004

[2] http://www.rocq.inria.fr/imedia/belga-logo.html

KNN Search for Image Queries



KNN search in SIFT feature space for image corpus & queries [1]

- Preprocessed feature vectors for corpus images
- Extraction of feature vectors for query images/subimages [2]
- High dimensional feature space (long feature vectors)
- Similarity score, correlation or distance function over the space
- KNN search to locate close matches for further classification

[1] D. G. Lowe, Inter. J. Comp. Vis., 2004

[2] http://www.rocq.inria.fr/imedia/belga-logo.html

Fast KNN Search: Other Applications

Quote *The computation of the nearest neighbor for the purpose of feature matching is the most time-consuming part of the complete recognition and localization algorithm.*

P. Azad, IROS, 2009

Fast KNN search will expedite

- ▷ Video segmentation M. Cooper, IEEE Trans. Multimedia, 2007
- ▷ Collaborative filtering X. Luo et al., Inter. J. Digit. Content Tech. Appl., 2011
- ▷ Image-data retrieval A. Joly and O. Buisson, ACM Multimedia 2009; P. Azad et al., IROS 2009
- ▷ GIS-moving objects in road networks C. Shahabi et al., SIGSPATIAL GIS, 2002
- ▷ Network intrusion detection L. Kuang and M. Zulkernine, ACM SAC, 2008
- ▷ Text categorization S. Manne et al., Inter. J. Comp. Appl., 2011

Outline

The KNN Search Problem

Problem Statement

To each and every query, locate k nearest neighbors, according to a score function, among n corpus data points in a d-dim space

d: the dimensionality of the search space
such as the length of the SIFT feature vectors

n: the number of corpus data points to query from

q: the number of query points

k: the number of nearest neighbors to locate for each query

Outline

State-of-the-Art Solutions

Typical solution components

- ▷ Search hierarchy for rapid elimination of far neighbors
  - ≻ Kd-trees [3], Balltrees [4], Metric trees [5]
  - ≻ Total # of comparisons :
    linear in k and sub-linear in global corpus size N, e.g., $O(\log N)$
- ▷ Exact KNN search in a corpus of reduced size n
  - ≻ linear in k and n
- ▷ Approximate KNN search
  - ≻ Locality-sensitive hashing [6]

---

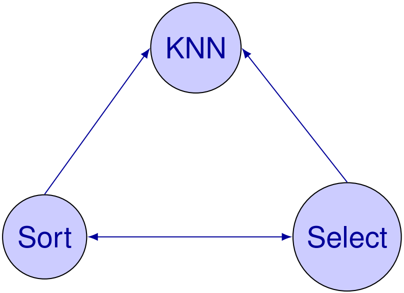
[3] J. L. Bentley, Comm. ACM, 1975

[4] S. Omohundro, Inter. Comp. Sci. Inst., TR, 1989

[5] J. Uhlmann, Info. Proc. Lett., 1991

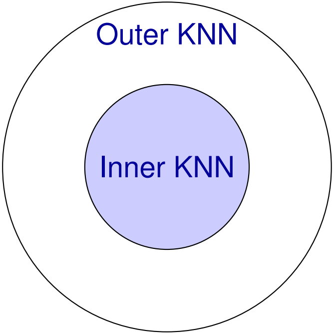
[6] P. Indyk, 30-th ACM STOC, 1999

Sort-Select-KNN Triangle



- sort-based KNN
- select-based KNN
- mutual connection between sort and select
- devil in algorithm and architecture detail

Inner and Outer KNNs



- Outer KNN
  - massive corpus data
  - quick reduction or decomposition
  - exploit data sparsity or clusters with heap data structure and operations
- Inner KNN
  - reduced corpus
  - utilize hardware architectures
  - exploit multiple queries
  - exploit relationship between query and corpus data
  - fast multi-dimension array operations

State-of-the-Art Solutions

More to be desired

- ▷ Synchronization on SIMD/SIMT processors such as GPUs
- ▷ Response latency for a single query
- ▷ Throughput rate for multiple queries
- ▷ Autotuning of performance
- ▷ Benchmarking at different integration scopes

KNN Search on GPUs: some other works

DataSet	Alg	Speedup		Parameter range			
(references)		X	base	*n*	*d*	*k*	*q*
`kdd-cup` [7]	exact	50	`CPU`	262,144	65	7	12,000
`uci adult` [8]	exact	15	`ANN`	30,956	123	16	1,605
`inria holidays` [9]	exact	64	`ANN`	65,536	128	20	1,024
`nasa images` [10]	exact	2	`Sort`	120,000	254	32	*any*
`recom system` [11]	exact	160	`CPU`	80,000	256	100	*any*
`labelme` [12] [13]	aprox.	40	`lshkit`	100,000	512	500	any

---

[7] S. Liang et al., IEEE Symp. Web. Soc., 2010

[8] Q. Kuang and L. Zhao, ISCSCT, 2009

[9] V. Garcia et al., ICIP, 2010

[10] R. J. Barientos et al., Euro-Par, 2011

[11] K. Kato and T. Hosino, CCGRID, 2010

[12] `http://www.labelme.csail.mit.edu`

[13] J. Pan and D. Manocha, GIS, 2011

Outline

Performance Analysis: Qualitative Factors

I. Architecture independent

- complexity in comparisons
- longest dependency path/depth
- variation in concurrency breadth

II. Architecture dependent

- effective concurrency breadth and dependency depth
- data locality: computation-communication ratio
- synchronization cost on GPUs

How well do we know the architectural impact ***quantitatively*** *?*

Outline

Performance Assessment : Quantitative References

Explore the two-ways relationship between SORT and SELECT

- SORT $\Longrightarrow$ SELECT
  - select or truncate *after* a complete ascending sort
  - **truncated sort** :
    truncate as early as possible *during* an ascending sort process

    as reference landmarks for quantitative performance assessment, or even as competitive candidates

- SELECT $\Longleftarrow$ SORT

  (omitted from this talk)

Truncated Sort Algorithms : Brief Summary

Algorithm	**Serial**	**Parallel** (length)	**Truncation Approach**
BubbleSort [14]	nk	$k(\log n - \log k + 1)$	k reversal passes
InsertionSort	nk	$k(\log n - \log k + 1)$	length-k array
HeapSort	$n \log k$	$k(\log n - \log k + 1)$	max-heap of size k
MergeSort [15]	$n \log k$	$k(\log n - \log k + 1)$	elimination by “half”
QuickSort [12, 16]	nk	$k(\log n - \log k + 1)$	elimination by “half”
RadixSort [12, 13]	$n \log_r c$	$\log_r c$	reverse radix (MSB)
BitonicSort [17]	$n \log^2 k$	$\log k \log n$	length-k bitonic

$$1 \leq k \leq n$$

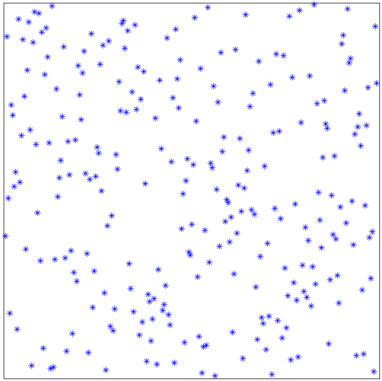
[14] C. E. Leiserson, Carnegie-Mellon Univ. Dep. of Comp. Sci., TR, 1979

[15] D. E. Knuth, The Art of Comp. Prog. 3, Addison-Wesley, 1973

[16] D. M. W. Powers, PACT, 1991

[17] K. E. Batcher, AFIPS, 1968

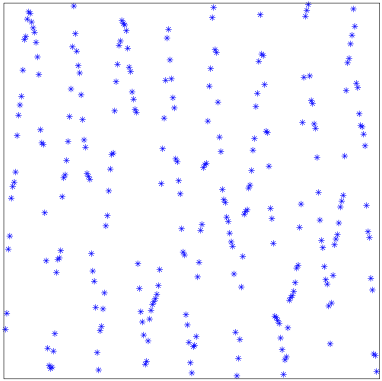
Quantitative Landmark : Truncated Bitonic Sort



- higher # pairwise comparisons
- inherently synchronous
  free of hashing or branching
- high data locality
  within practical range of k
- regular structures
  data access, program

A remarkable quantitative reference for KNN search performance on SIMD/SIMT processors

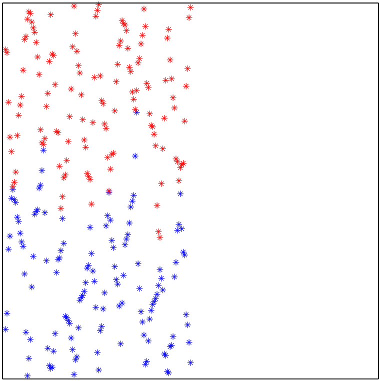
Quantitative Landmark : Truncated Bitonic Sort



- higher # pairwise comparisons
- inherently synchronous
  free of hashing or branching
- high data locality
  within practical range of k
- regular structures
  data access, program

A remarkable quantitative reference for KNN search performance on SIMD/SIMT processors

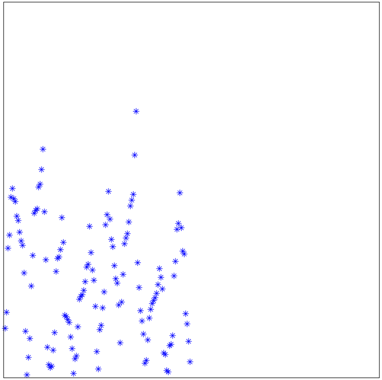
Quantitative Landmark : Truncated Bitonic Sort



- 👎 higher # pairwise comparisons
- 👍 inherently synchronous
  free of hashing or branching
- 👍 high data locality
  within practical range of k
- 👍 regular structures
  data access, program

A remarkable quantitative reference for KNN search performance on SIMD/SIMT processors

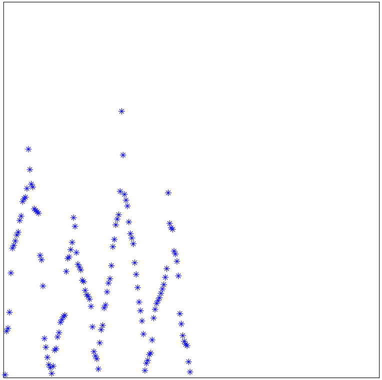
Quantitative Landmark : Truncated Bitonic Sort



- higher # pairwise comparisons
- inherently synchronous
  free of hashing or branching
- high data locality
  within practical range of k
- regular structures
  data access, program

A remarkable quantitative reference for KNN search performance on SIMD/SIMT processors

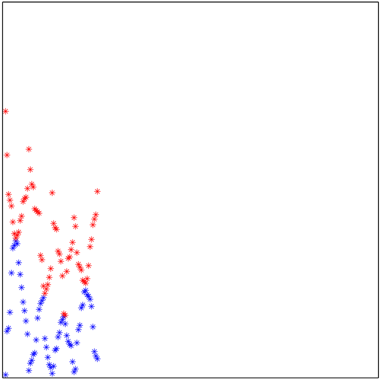
Quantitative Landmark : Truncated Bitonic Sort



- higher # pairwise comparisons
- inherently synchronous
  free of hashing or branching
- high data locality
  within practical range of k
- regular structures
  data access, program

A remarkable quantitative reference for KNN search performance on SIMD/SIMT processors

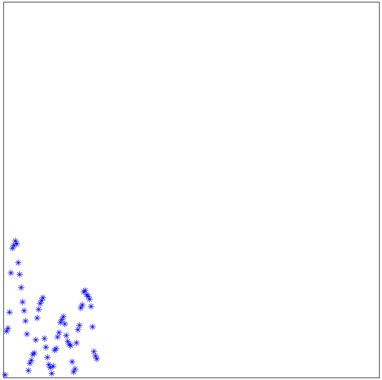
Quantitative Landmark : Truncated Bitonic Sort



- higher # pairwise comparisons
- inherently synchronous
  free of hashing or branching
- high data locality
  within practical range of k
- regular structures
  data access, program

A remarkable quantitative reference for KNN search performance on SIMD/SIMT processors

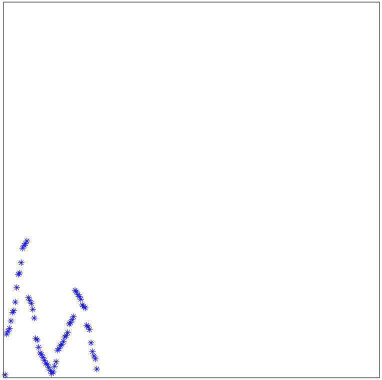
Quantitative Landmark : Truncated Bitonic Sort



- higher # pairwise comparisons
- inherently synchronous
  free of hashing or branching
- high data locality
  within practical range of k
- regular structures
  data access, program

A remarkable quantitative reference for KNN search performance on SIMD/SIMT processors

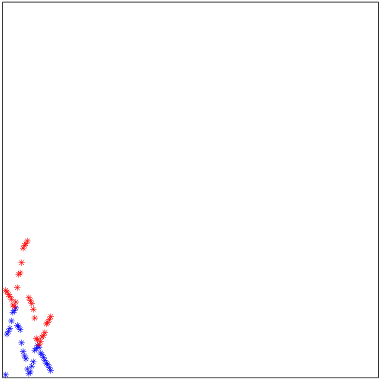
Quantitative Landmark : Truncated Bitonic Sort



- higher # pairwise comparisons
- inherently synchronous
  free of hashing or branching
- high data locality
  within practical range of k
- regular structures
  data access, program

A remarkable quantitative reference for KNN search performance on SIMD/SIMT processors

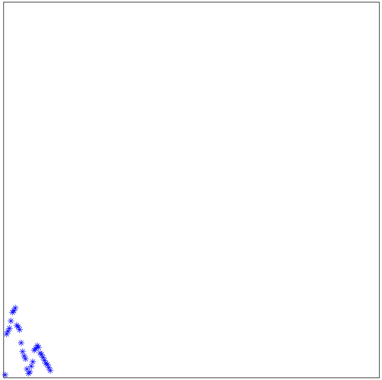
Quantitative Landmark : Truncated Bitonic Sort



- higher # pairwise comparisons
- inherently synchronous
  free of hashing or branching
- high data locality
  within practical range of k
- regular structures
  data access, program

A remarkable quantitative reference for KNN search performance on SIMD/SIMT processors

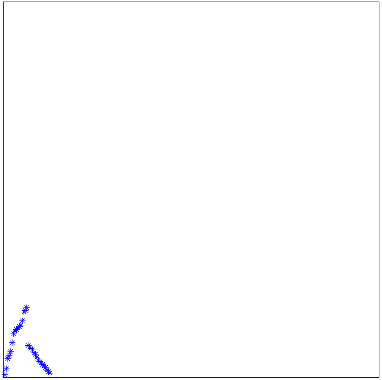
Quantitative Landmark : Truncated Bitonic Sort



- higher # pairwise comparisons
- inherently synchronous
  free of hashing or branching
- high data locality
  within practical range of k
- regular structures
  data access, program

A remarkable quantitative reference for KNN search performance on SIMD/SIMT processors

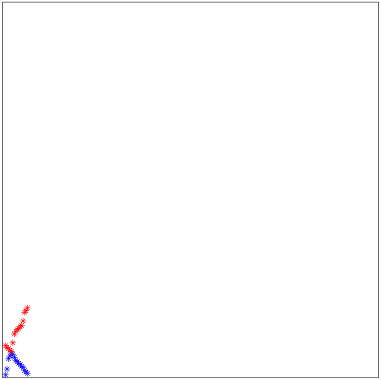
Quantitative Landmark : Truncated Bitonic Sort



- higher # pairwise comparisons
- inherently synchronous
  free of hashing or branching
- high data locality
  within practical range of k
- regular structures
  data access, program

A remarkable quantitative reference for KNN search performance on SIMD/SIMT processors

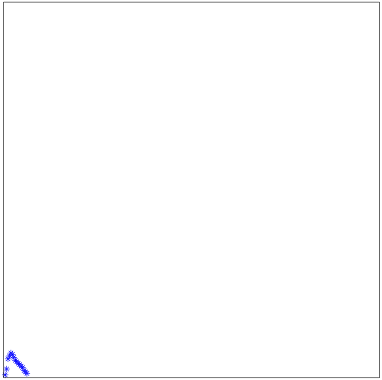
Quantitative Landmark : Truncated Bitonic Sort



- higher # pairwise comparisons
- inherently synchronous
  free of hashing or branching
- high data locality
  within practical range of k
- regular structures
  data access, program

A remarkable quantitative reference for KNN search performance on SIMD/SIMT processors

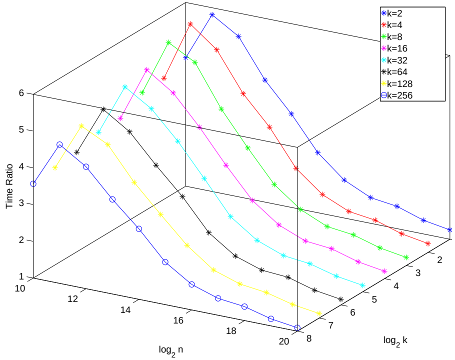
Quantitative Landmark : Truncated Bitonic Sort



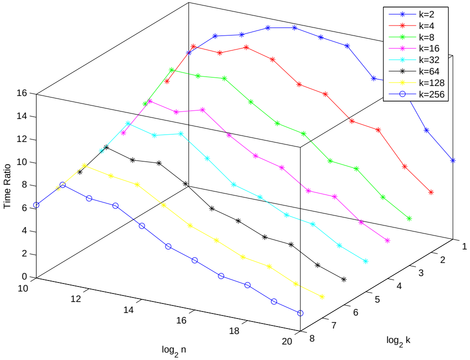
- higher # pairwise comparisons
- inherently synchronous
  free of hashing or branching
- high data locality
  within practical range of k
- regular structures
  data access, program

A remarkable quantitative reference for KNN search performance on SIMD/SIMT processors

THRUST::SORT vs Truncated Bitonic Sort

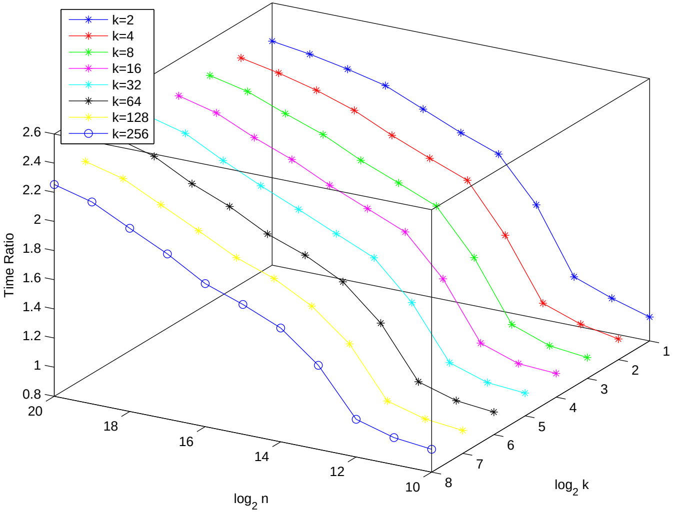


Inclusion of Score Evaluation

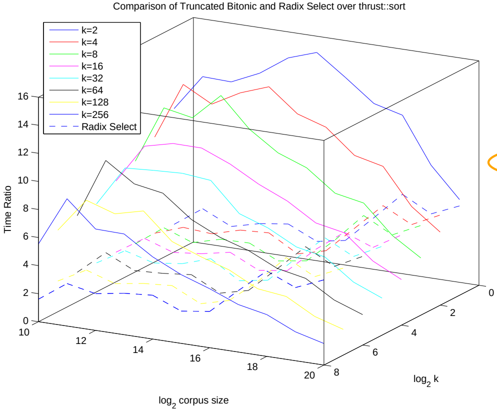


Exclusion of Score Evaluation

Truncated Sorting Interleaved with Scoring



Truncated BitonicSort & MGPU RadixSelect [18]



Manifest of Synch. Cost

Truncated Bitonic Sort substantially outperforms MGPU Radix Select over the effective range

Here, `thrust::sort` used as a common base for comparison

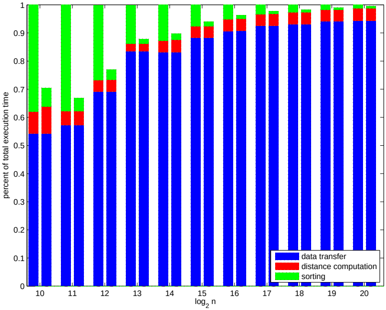
[18] www.moderngpu.com

Outline

KNN Search in Multistage Streaming on GPUs

- transporting and buffering large corpus data in batches (batch size n)
- merging KNNs between the previous and the current corpus batches
- inclusion of score evaluation and pre/post computation tasks (separated or interleaved)
- multiple queries (as desirable in certain applications)

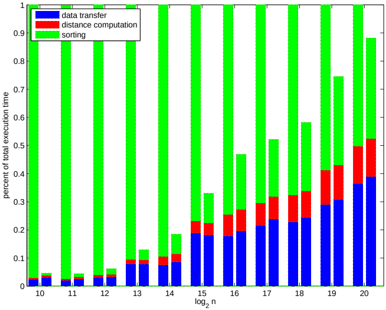
MultiStage KNN Profile on GPUs: Single Query



Profile in total execution time

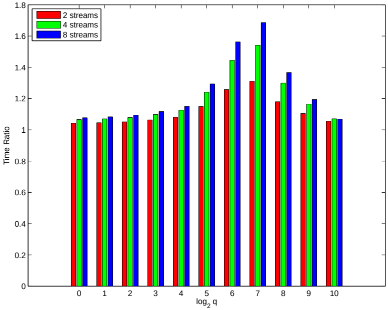
- Left bars: Truncate after sorting using `thrust::sort` in percentile: data transfer dominant when the batch size *n* is large
- Right bars: Truncated Bitonic normalized against the left bars

KNN Search Profile on GPUs: Multiple Queries



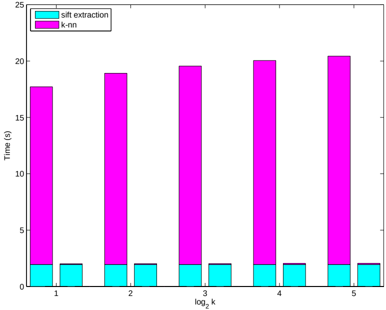
- Left bars: Truncate after sorting using `thrust::sort`
- Right bars: Truncated Bitonic normalized against the left bars

KNN Search in Multistage Streaming on GPUs



- $16,777,216$ vectors of 128 dimensions
- Overlapping data transfer and computations
- Up to $\times 1.75$ speed-up from using only 1 GPU stream

SIFT Feature Matching :



- VLFeat, a CV Library [a]
  - sequential implementation of feature extraction (with SIFT) and KNN search [b]
  - approximate k-NN using tree space partition
- Speed-up over VLFeat
  - 60X with 128 queries
  - 180 $\sim$ 250X with 512 queries

[a] http://www.vlfeat.org

[b] Parallel SIFT vector extraction available on GPUs: http://www.cs.unc.edu/ ccwu/siftgpu/

Summary

We have

- ▷ addressed response latency & throughput issues
- ▷ explored the SORT-SELECT relationship
- ▷ exposed the synchronization cost on GPUs & provided references for quantitative performance assessment
  (relevant for approximate KNN search as well)
- ▷ suggested options and opportunities to better exploit GPUs for rapid KNN search queries
- ▷ codes and test data available at `http://autogpu.ee.auth.gr`

Acknowledgments

NVIDIA academic research equipment support

Marie Curie International Reintegration Program, EU

National Science Foundation (CCF), USA