

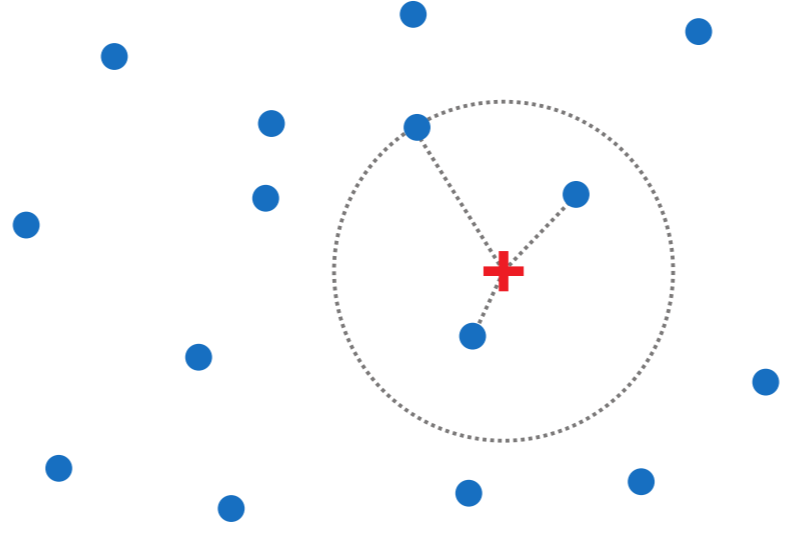
KNN Search: fast GPU-based implementations and application to high-dimensional feature matching

Vincent Garcia, Éric Debreuve, Frank Nielsen, Michel Barlaud

1. INTRODUCTION

KNN SEARCH PROBLEM DEFINITION

- Let \mathcal{R} be a set of m reference points $\mathcal{R} = \{r_1, r_2, \dots, r_m\}$ defined in \mathbb{R}^d
- Let \mathcal{Q} be a set of n query points $\mathcal{Q} = \{q_1, q_2, \dots, q_n\}$ defined in \mathbb{R}^d
- KNN search problem: for each $q \in \mathcal{Q}$, determine the k points closest to q among \mathcal{R}



- Distance between points: usually Euclidean but can be anything
- **Highly demanding in terms of computation time**
- Applications: 3D object rendering, content-based image retrieval, statistics, biology ...

2. BRUTE-FORCE ALGORITHM

ALGORITHM

Let $q \in \mathcal{Q}$ be a query point

1. Compute the distance between q and the m reference points of \mathcal{R}
2. Sort the m distances

The k -nearest neighbors of q are the k points of \mathcal{R} corresponding to the k lowest distances. The output of the algorithm can be the ordered set of these k distances, the set of the k neighbors (their indices in \mathcal{R}) ordered by increasing distance, or both

PROPERTIES

- High-complexity: $O(nmd)$ multiplications for the $n \times m$ distances computed and $O(nm \log m)$ for the n sorting processes
- Algorithm highly-parallelizable = perfectly suitable for a GPU implementation

3. DISTANCE COMPUTATION

DISTANCE BETWEEN TWO POINTS

- Let x and y be two points in \mathbb{R}^d . The Euclidean distance is given by: $\rho(x, y) = \sqrt{\sum_{i=1}^d (x_i - y_i)^2}$.
- Squared distance computation can be rewritten to involve matrix additions and multiplications:

$$\rho^2(x, y) = \|x\|^2 + \|y\|^2 - 2x^\top y$$

DISTANCE BETWEEN TWO SETS OF POINTS

- Let R be a matrix of size $d \times m$ containing the m reference points
- Let Q be a matrix of size $d \times n$ containing the n query points
- The $m \times n$ -matrix $\rho^2(R, Q)$ containing all the pairwise squared distances between query points and reference points is given by

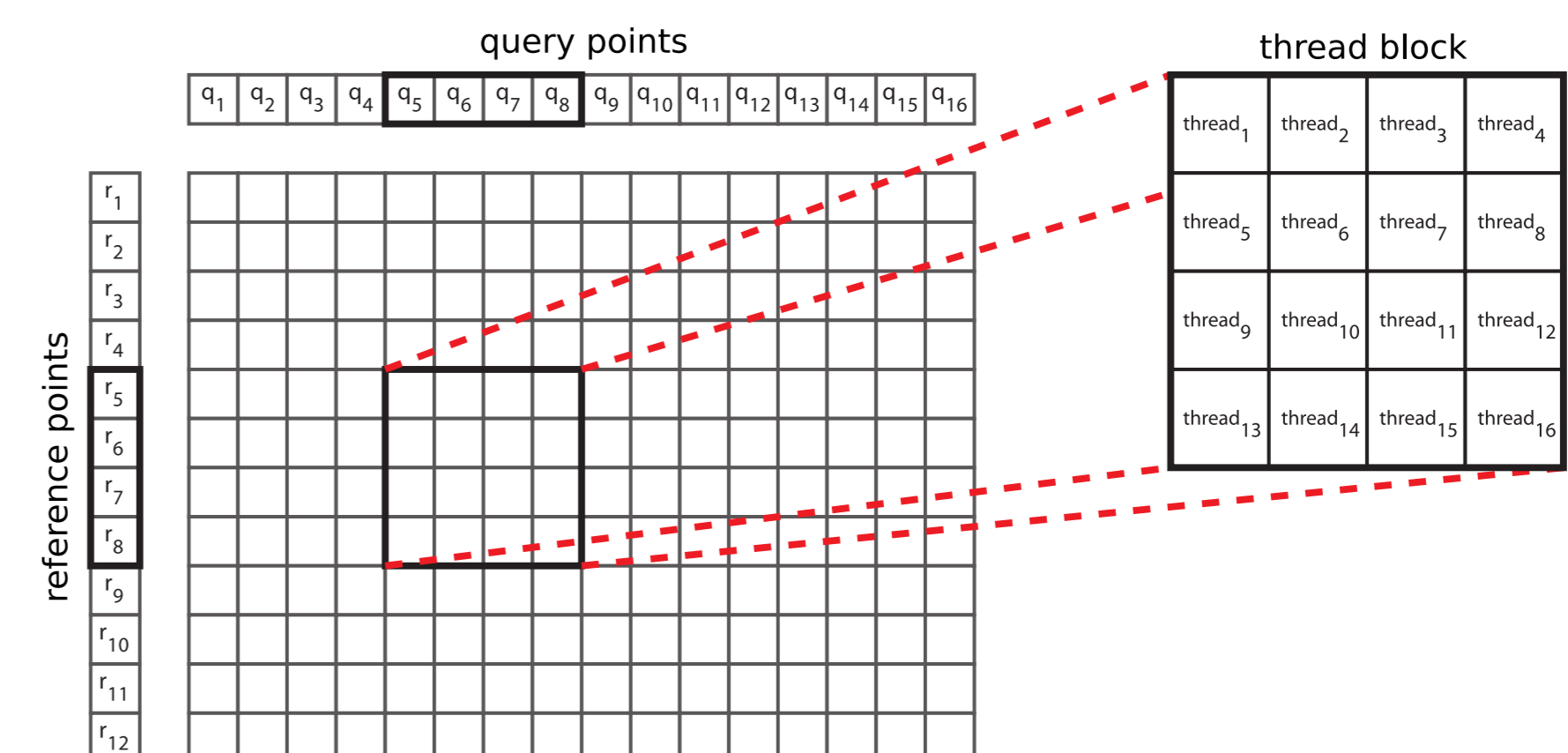
$$\rho^2(R, Q) = N_R + N_Q - 2R^\top Q$$

- The elements of the i^{th} row of N_R are all equal to $\|r_i\|^2$
- The elements of the j^{th} column of N_Q are all equal to $\|q_j\|^2$

4. CUDA IMPLEMENTATION

1. Compute the distance matrix of size $m \times n$ containing the distances between the n query points and the m reference points. The computation of this matrix was fully parallelized: each thread computed the distances between one query point and one reference point.
2. Sort the distance matrix. The n sorting processes were parallelized: each thread sorted all the distances computed for a given query point.

The KNN correspond to the k lowest distances (uppermost $k \times n$ -submatrix of the sorted matrix).



5. CUBLAS IMPLEMENTATION

1. Compute N_R using CUDA
2. Compute N_Q using CUDA
3. Compute the $m \times n$ -matrix $A = -2R^\top Q$ using CUBLAS
4. Compute $B = N_R + A$ using CUDA
5. Sort in parallel each column of B (with n threads); The resulting matrix is denoted by C
6. Compute $D = N_Q + C$ using CUDA
7. Compute the square root of the first k elements of D to obtain the k smallest distances (coalesced read/write); The resulting matrix is denoted by E

The KNN correspond to the k lowest distances (uppermost $k \times n$ -submatrix of E)

Implementations available at:
www.i3s.unice.fr/~creative/KNN/

6. EXPERIEMENTS

SETUP

- Dell Precision M6400: Intel Core 2 duo/2.53GHz, 4Go DDR2, NVIDIA Quadro FX 3700M
- Microsoft Windows XP 32 bits, NVIDIA CUDA 2.2, and Matlab 2007b

KNN SEARCH ON SYNTHETIC DATASETS

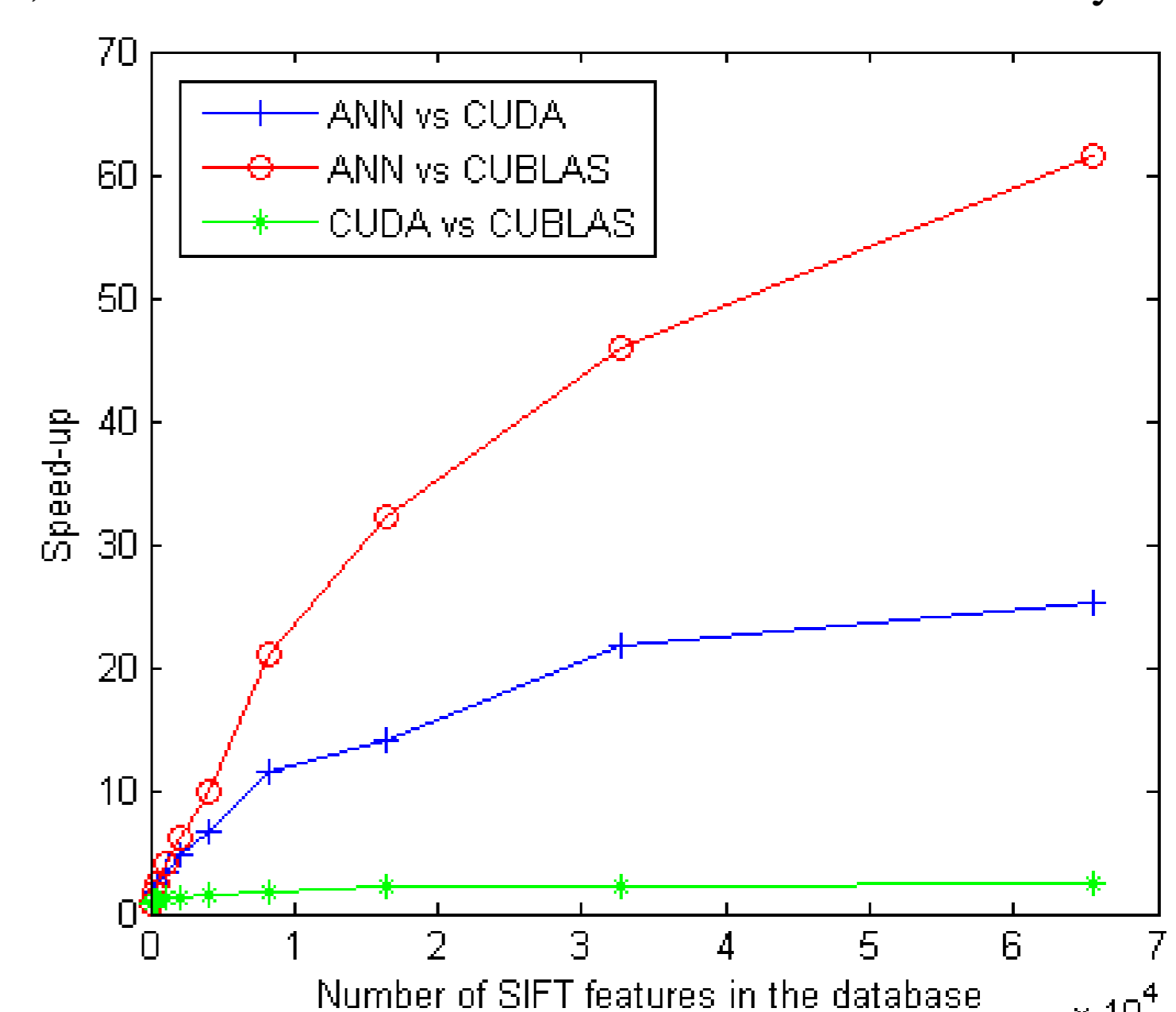
- Points (\mathcal{R} and \mathcal{Q}) were randomly drawn from $\mathcal{N}(0, 1)$
- n = number of points (reference and query), d = dimension of the points, $k = 20$
- ANN refers to the fast C++ ANN library

	Method	n=256	n=512	n=1024	n=2048	n=4096	n=8192	n=16384	n=32768	n=65536
d=1	ANN	0.001	0.002	0.004	0.007	0.015	0.031	0.063	0.132	0.277
	CUDA	0.042	0.046	0.054	0.063	0.081	0.154	0.480	2.489	8.302
	CUBLAS	0.042	0.047	0.055	0.066	0.091	0.213	0.698	1.615	5.694
d=4	ANN	0.003	0.005	0.012	0.027	0.059	0.125	0.275	0.591	1.364
	CUDA	0.042	0.048	0.055	0.066	0.086	0.176	0.561	2.591	8.425
	CUBLAS	0.042	0.048	0.057	0.068	0.093	0.220	0.733	1.812	6.076
d=16	ANN	0.007	0.028	0.109	0.385	1.421	5.468	20.289	84.503	378.496
	CUDA	0.043	0.049	0.056	0.070	0.105	0.247	0.805	2.542	8.900
	CUBLAS	0.044	0.049	0.056	0.067	0.092	0.203	0.791	2.123	7.225
d=64	ANN	0.017	0.073	0.299	0.949	3.279	13.365	74.183	313.527	1296.367
	CUDA	0.044	0.050	0.062	0.087	0.176	0.528	1.950	5.518	20.441
	CUBLAS	0.044	0.049	0.057	0.069	0.102	0.242	0.904	3.104	10.887
d=256	ANN	0.051	0.194	0.742	2.933	14.579	76.454	334.509	1053.819	3559.731
	CUDA	0.045	0.055	0.081	0.159	0.459	1.641	6.910	19.381	75.718
	CUBLAS	0.044	0.050	0.060	0.079	0.146	0.405	2.394	7.157	29.480

- CUDA was up to 64X than ANN
- CUBLAS was up to 189X faster than ANN
- CUBLAS was up to 4X faster than CUDA

KNN SEARCH APPLIED TO SIFT MATCHING

- SIFT: point in \mathbb{R}^{128}
- SIFT matching: application to CBIR
 - SIFT features extracted from a set of reference images and stored in a database
 - SIFT features extracted from a query image I
 - For each SIFT of I , find the k closest features in the database (KNN search)
 - A voting algorithm enables to find the images most similar to I in the image database
- For this experiments, SIFT features were extracted from INRIA Holidays dataset



- CUDA was up to 25X faster than ANN
- CUBLAS was up to 62X faster than ANN
- CUBLAS was up to 2.5X faster than CUDA