Also publ. in: General Purpose Computation on Graphics Hardware Workshop: Methods, Algorithms and Applications (ICCS), 2006

A Graphics Hardware Accelerated Algorithm for Nearest Neighbor Search

Benjamin Bustos*, Oliver Deussen, Stefan Hiller, and Daniel Keim

Department of Computer and Information Science, University of Konstanz
{bustos,deussen,hiller,keim}@informatik.uni-konstanz.de

Abstract. We present a GPU algorithm for the nearest neighbor search, an important database problem. The search is completely performed using the GPU: No further post-processing using the CPU is needed. Our experimental results, using large synthetic and real-world data sets, showed that our GPU algorithm is several times faster than its CPU version.

1 Introduction

Recent publications have studied the usage of GPUs as co-processors for database applications. Not surprisingly, the first papers mainly focused on graphics related operations in spatial databases, e.g., methods to accelerate the refinement step of spatial selections and joins using the GPU [1] or how to integrate the hardware acceleration provided by GPUs with a commercial DBMS for spatial operations [2]. Govindaraju et al. [3] focused on general database operations on the GPU.

An important, but challenging, database problem is the nearest neighbor (NN) search. The NN of a given query point $q \in \mathbb{R}^d$ in the database $\mathbb{D} \subset \mathbb{R}^d$ is defined as $u_{NN} = \{u' \in \mathbb{D} \mid \forall u \in \mathbb{D}, u \neq u' : \delta(u', q) \leq \delta(u, q)\}$ for a given distance function δ. Finding the NN has many applications in fields like similarity search in multimedia databases, data mining, and information retrieval. Several indexing methods have been proposed for implementing NN search [4]. However, most of the experiments reported in this area show that the performance of a linear scan is highly competitive for high-dimensional data sets, and that it can be faster than any index structure in such spaces. In addition, the famous results by Beyer et al. [5] have shown that theoretically, for very high dimensionality, the NN problem is inherently linear for a wide range of data distributions.

In this paper, we provide a GPU implementation of the linear scan based NN search algorithm. We evaluate our solution using large real and synthetic data sets, obtaining significant speed ups over the CPU-based algorithm.

2 GPU Implementation of Nearest Neighbor Search

A linear scan based NN search computes the distance δ between a query object $q \in \mathbb{R}^d$ and all objects in the database $\mathbb{D} \subset \mathbb{R}^d$. The object with minimum

* On leave from the Department of Computer Science, University of Chile.

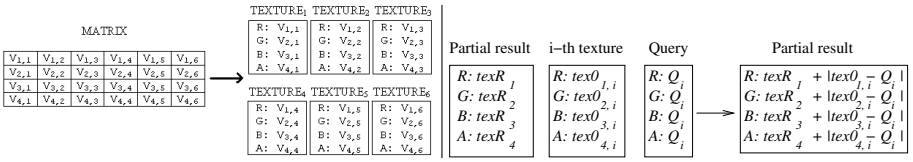

Fig. 1. Data organization in textures (left), and how does FP1 work (right).

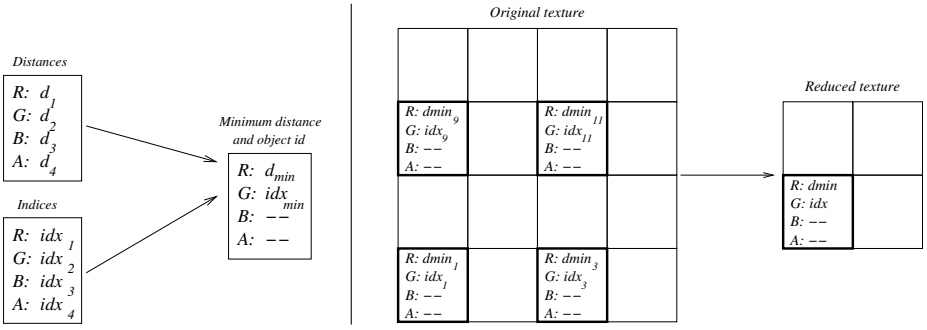

Fig. 2. Texel processing by FP2 (left) and texture reduction by FP3 (right).

distance to q is returned as the NN. We depict now how to efficiently implement this algorithm with the aid of a GPU. The first step of the algorithm is to load the vectors into the graphics card texture memory. For this purpose, we create d 2-D textures. Each of them stores one coordinate value of all vectors. As we use the four color channels to store the data, each texel from the i^{th} texture contains the i^{th} coordinate value of four different vectors (see Figure 1 (left)).

We use three fragment programs (FPs) to implement the linear scan. The first FP computes the distance between each object $u \in \mathbb{D}$ and the query q. As distance δ, we use the *Manhattan distance* (other metrics are possible). To fully exploit the potential of the GPU, we compute the difference between coordinates for several dimensions simultaneously. At each pass, the algorithm processes t textures (dimensions) in parallel, for a total of d/t passes. The results from previous iterations are aggregated with the results of the current pass in an additional texture tex_R (which initially contains zeros). Figure 1 (right) illustrates.

The next rendering pass determines the NN to q. The second FP computes the minimum distance value within the color+alpha channels (d_{min}) and associates this distance value to the index of its correspondent object (idx_{min}) (see Figure 2 (left)). The FP compares the four values stored in each texel, keeping the minimum value in the red channel and storing its associated index (an integer value between 1 and n) in the green channel. The third FP searches the minimum distance between four appropriately selected texels, and iteratively reduces the texture size by a factor of 4 at each pass. The minimum distance and its associated index are stored in the red and green channel, respectively (see Figure 2 (right)). This iterative reduction is the tricky part in the search

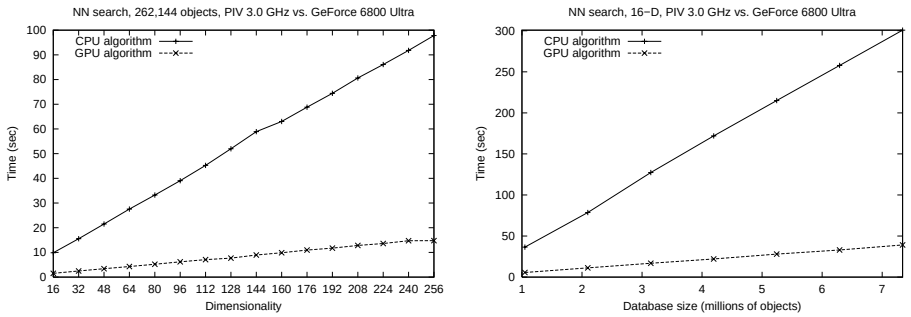

Fig. 3. Results varying dimensionality (left) and database size (right).

algorithm, and it is simulated by storing the results on the first quarter of the original texture, and only this quarter is used at the next rendering step. The algorithm stops when the texture has been reduced to size 1×1. Then, only one texel is read back from the graphics card memory. The object whose index was stored in the retrieved texel is returned as the NN of q.

3 Experimental Results

We used an NVIDIA GeForce 6800 Ultra graphics card with 256 MByte of memory. The CPU is a Pentium IV 3.0 GHz. The CPU algorithm was implemented in C++ and compiled with the Intel C++ Compiler v8. We activated all optimization flags for producing the fastest possible SSE2 enhanced CPU code. We used the Cg compiler v1.3 for the GPU FPs. We measured query upload time, computation time, and texture download time (the databases are uploaded only once into the graphics memory, thus this upload time is amortized over the queries).

The synthetic database consisted of 262,144 vectors (16-D to 256-D), with random coordinates values uniformly distributed in the range $[0.0, 1.0]$. We averaged the results over 1,000 random query vectors, and we got the best times using $t = 8$. Figure 3 (left) shows the obtained results: The GPU algorithm achieved an average speed-up factor of 6.4x. Our algorithm also scaled well when using different database sizes. If the data did not fit into one texture, we partitioned the data in blocks of about 1 million objects and run the algorithm on each block iteratively. Figure 3 (right) shows the results for 1 to 7.5 million vectors.

We also tested our GPU algorithm using real-world databases. The first one is the *Forest CoverType* (*UCI-KDD-A*) which contains data about different forest cover types obtained by the U.S. Forest Service (54-D, 250,000 observations). The second one is the *Corel image features* (*UCI-KDD-B*), which contains features of images extracted from a Corel image collection (32-D, 65,000 images). Both sets are available at the *UCI KDD Archive* [6]. The third one is a 3D CAD database (512-D, 16,000 models). For each data set, we selected 1,000 random objects as queries for the NN search. Figure 4 (left) shows the results. We observed speed-up factors of, respectively, 6.4x, 4.5x, and 4.2x over the CPU algorithm.

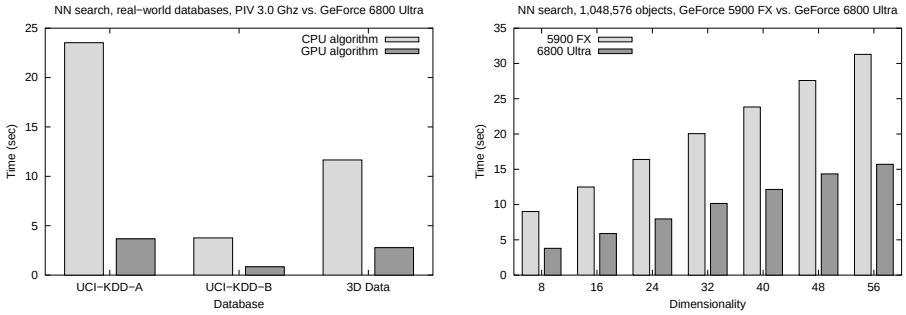

Fig. 4. Results using real data sets (left) and comparison between two GPUs (right).

Finally, we compared the GeForce 6800 Ultra card with one card from the previous generation, namely the GeForce 5900 FX, to estimate what kind of improvements one could expect for the future. With regard to hardware (pixel shaders and memory bandwidth), the GeForce 6800 Ultra should be at least twice as fast as the GeForce FX 5900. Figure 4 (right) shows the obtained results. For the next generation of GPUs, we expect a similar speed-up factor.

4 Conclusions

We presented a GPU accelerated algorithm for NN search in high-dimensional vector spaces. We described how to map vectors into texture data, without restrictions on the dimensionality of the data, and we showed that relatively simple FPs (including a texture reduction process) can efficiently return the NN object. Experimental results using synthetic and real-world data sets showed that our GPU algorithm provide a significant speed improvement over the CPU algorithm, with linear scalability in dimensionality and database size.

References

1. Sun, C., Agrawal, D., Abbadi, A.: Hardware acceleration for spatial selections and joins. In: Proc. ACM Intl. Conf. on Managment of Data. (2003) 455–466
2. Bandi, N., Sun, C., Abbadi, A., Agrawal, D.: Hardware acceleration in commercial databases: A case study of spatial operations. In: Proc. Intl. Conf. on Very Large Databases, Morgan Kaufmann (2004) 1021–1032
3. Govindaraju, N., Lloyd, B., Wang, W., Lin, M., Manocha, D.: Fast computation of database operations using graphics processors. In: Proc. ACM Intl. Conf. on Managment of Data. (2004) 215–226
4. Böhm, C., Berchtold, S., Keim, D.: Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. ACM Computing Surveys **33**(3) (2001) 322–373
5. Beyer, K., Goldstein, J., Ramakrishnan, R., Shaft, U.: When is "nearest neighbor" meaningful? In: Proc. 7th Intl. Conf. on Database Theory. (1999) 217–235
6. Hettich, S., Bay, S.: The UCI KDD archive [http://kdd.ics.uci.edu] (1999)