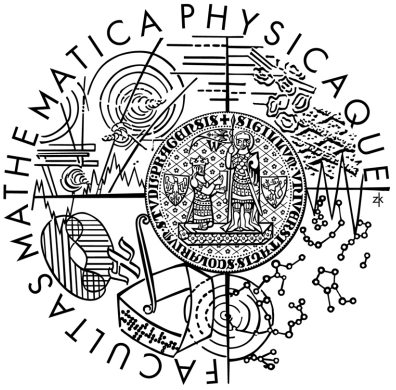
Charles University
Faculty of Mathematics and Physics

# HABILITATION THESIS




**Martin Kruliš**

# GPU-Accelerated Methods for Content-based Retrieval

*Computer Science, Software Systems*

Prague, Czech Republic 2019

# Contents

# Preface

This thesis presents results from a research that aimed at accelerating selected tasks in multimedia databases by employing modern general-purpose GPUs. This research track has been initiated during my graduate studies and its main part was carried out during my employment at the Department of Software Engineering, Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic.

The selected topics cover multiple areas, especially the evaluation of similarity distance functions, which is an essential part of content-based retrieval process in multimedia databases, indexing, approximative search, and image preprocessing. In all cases, the main objective was to increase performance (i.e., reduce time and improve efficiency) of selected tasks by offloading essential parts of the algorithms to GPU devices. Presented contributions also provide valuable insights which may be generalized for accelerating similar database and image processing problems. This thesis comprise published research papers that summarize our contributions in selected areas and connecting comments which explain the contributions in broader context.

Substantial portion of the presented research was conducted as a part of my Czech Science Foundation grant P103-14-14292P: *Employing modern parallel architectures in specific domains of database systems*. The research is of a collective rather than individual in nature. However, I would like to declare that the design and implementation of GPU algorithms and parallelization methods are mine contributions.

I would like to thank to my coauthors and fellow researchers both from my department and foreign collaborators from RWTH Aachen, Germany, and University of Geneva, Switzerland, namely: David Bednárek, Christian Beecks, Michal Brabec, David Honzátko, Steffen Kirchhoff, Jakub Lokoč, Stéphane Marchand-Maillet, Hasmik Osipyan, Thomas Seidl, Tomáš Skopal, Jakub Yaghob, and Filip Zavoral.

Martin Kruliš

# CHAPTER 1

# Introduction

Achieving higher computing performance and better efficiency is important. It positively impacts many scientific fields and invites new opportunities as we have witnessed recently in the domain of deep learning. It also plays essential role in lowering carbon dioxide emissions and stopping global warming[1]. In the past, we have relied heavily on hardware improvements as each new generation of processors brought higher performance and better power efficiency. Famous observations like the *Moore's law*, *Dennard scaling*, or the *Koomey's law* lulled the programmers into false sense of confidence that the code optimizations are less important as the hardware development will solve most of the performance problems. Unfortunately, these laws are dead [51, 54].

One of the trends in hardware development that prevailed is parallelism. In fact, it becomes even more important than ever since it opened ways for releasing tremendous computational power utilizing existing technologies for processor design and construction whilst keeping the electrical consumption and the complexity of the chips reasonable [21]. Multi-core parallelism was introduced in the mainstream CPUs at the dawn of this century and it presently occupies all levels of products from small handheld devices to large servers. Furthermore, massively parallel platforms like general-purpose GPUs have expanded beyond their original intent of use and found their way to various domains [47], especially in the realm of soft-computing [15, 31].

Perhaps the greatest drawback of parallel computing is that the programmers are not ready to take the advantage of the hardware and to utilize available processing units to their full potential. Despite the fact that the support for parallel programming in languages, frameworks, and libraries improved significantly in the past decade, most computer science domains are still reluctant to fully embrace parallel processing. The main reason for this resistence is that the parallelism brings another level of complexity in the code design. Therefore, it requires special skills which most programmers do not posses and it increases probability of introducing additional bugs and errors into the code.

[1] Our attention lies in the direction of useful computations. Power-wasting applications like the block-chain verification are also a great concern, but they cannot be helped by improving efficiency.

These issues are even more pronounced in case of general-purpose GPUs. GPU architectures are quite different from regular CPU architectures. First of all, a GPU comprise thousands of computing cores so the programmer has to choose a much finer-grained parallelism and avoid explicit synchronization. Furthermore, the cores are tightly coupled as they share resources such as computing units, caches, or even instruction schedulers. This design promotes data parallelism paradigm where multiple data elements are being processed by the same routine. Unfortunately, other paradigms like the task parallelism, where independent tasks (i.e., routines) are processed concurrently, have only limited support. Finally, the GPUs are much more sensitive to data layout since they contain much smaller caches with respect to the number of cores. Hence, even small changes in data organization, alignment, or access patterns may have profound impact on overall performance and it requires extensive coding experience as well as detailed knowledge of the hardware to design algorithms and data structures in a way that is optimal for GPUs.

It is our job as computer scientists to explore new ways and methods that will allow streamlining utilization of parallel architectures in everyday computing so that regular programmers may harness the computational power of contemporary hardware more easily. This thesis contributes to the effort by addressing performance issues of multimedia databases with particular focus on image datasets such as photo galleries. These databases are very interesting from research point of view for several reasons:

- Multimedia databases employ novel approaches to data querying like content-based retrieval and similarity search. Furthermore, these queries are typically initiated by end-users so they need to be processed speedily to ensure reasonable interactivity.
- Unlike relational databases, multimedia databases have to process much more complex objects (e.g., images), which requires much higher computational power. Therefore, persistent data storage is no longer the most common bottleneck of these database systems and there are many opportunities for parallel accelerators.
- Multimedia databases have the potential to grow very fast as they are often connected to social networks and other massively used web applications. Querying these datasets in real time requires combining efficient processing methods with modern parallel hardware.

The main objective is to propose novel GPU algorithms to accelerate the most time-critical tasks in multimedia databases. Accelerated algorithms should lower the latency of queries and improve efficiency of other database operations such as index construction or data preprocessing. Although we cannot address every existing database problem individually, the selected problem representatives cover all essential operations and provide insights which should simplify introducing GPUs into database systems in general. The following issues have been addressed and solved in our research.

First, we focused on accelerating the **query evaluation** process in the similarity models. When a similarity query is being evaluated, the most time is typically spent by computing similarity (distance) functions between the query and all database objects, or at least all candidate objects yielded by an index. Even though it might seem that GPU

implementation of this problem is straightforward as it already follows the data-parallel paradigm, there are several issues that needs to be addressed. First of all, the size of the object descriptors (i.e., the inputs) and the spatial requirements of some functions (like the Signature Quadratic Form Distance [12]) cannot be easily accommodated by memory resources of single GPU core. Furthermore, if indexing is employed, there may not be enough object candidates (i.e., similarity distances) available to occupy all GPU cores simultaneously. Due to these issues, a much finer-grained approach to parallelism and appropriate memory organization has to be selected. Section 2.1, we describe the details and parallelization challenges of two selected similarity functions – the Signature Quadratic Form Distance [12] and the Levenshtein's Edit Distance [30]. The contributions are presented in Chapters 3 and 4.

Employing **metric indices with parallel distance computations** on GPUs raised one rather specific issue. The nearest neighbour ($k$NN) queries accelerated by metric indexing methods often rely on sequential updates of the intermediate top-$k$ set for optimal effectiveness which limits opportunities for computing the distances in parallel. To solve this problem, we have devised an estimation method that fully exploits parallelism whilst maintaining a high selectivity of the index without frequent sequential updates of the intermediate result set. The perils of employing metric indexing with parallel processing are addressed in Section 2.2 and the contribution is summarized in Chapter 5.

In some use cases, the speed of object retrieval is much more important than the accuracy. In such scenarios, an **approximative indexing** methods are often used. We have focused on a method called permutation-based indexing [9, 22, 42] applied on high dimensional Euclidean spaces. Index utilization relies on the same principles already addressed in Section 2.1, so we focus mainly on the acceleration of index construction. Having an efficient construction process allows us to index the data as the dataset grows and to re-index the data to achieve better recall if the dataset changes. The permutation-based indexing and its challenges are described in Section 2.3 while Chapters 6 and 7 summarize our contributions.

Similarity models operate on object descriptors which are extracted from database object contents. Even though these descriptors can be cached easily in most cases, it still stands to reason to accelerate their extraction by the means of GPUs. We have used image **feature signatures** in our experiments with SQFD; thus, we also accelerated their extraction process. The selected feature signatures comprise localized color information as well as texture details [44]. They are also adaptive – i.e., they represent large monotonous parts of an image with fewer features and important details with more features. The issues of the extraction are addressed in Section 2.4 and Chapter 8 presents the actual contributions.

Finally, we have focused on the topic of data preprocessing – particularly **image denoising** which improves the quality of the stored images. We have accelerated Block-Matching 3D Filtering [18] denoising method which employs image self-similarity to improve traditional threshold filtering along with 3D decorrelating transformation of the image data. Even though this is not a typical database-related problem (the denoising can be used on individual images as well as on the entire database), it is related to our research since it is based on similarity of image patches and it provides valuable insights to accelerating various image processing subroutines. Furthermore, it improves data quality which may be

an issue for some datasets like photo galleries. The BM3D method and its parallelization challenges were described in Section 2.5 and the contributions are summarized in Chapter 9.

It is necessary to point out, that some of the presented methods have been outperformed in the terms of effectivity since this work has started. For instance, similarity search in image databases is currently dominated by methods based on convolutional neural networks [13]. On the other hand, current success of neural networks is contributed to the fact that they can be trained very quickly on modern parallel hardware, especially GPUs [15], and our contributions aimed at improving efficiency by the means of parallel processing, not at improving effectivity of existing methods. Furthermore, most observations made in the presented papers may be generalized of similar tasks and even though the presented methods have been designed primarily for the image databases (e.g., photo galleries), both the distance functions and the indexing methods are applicable for other types of databases, such as spatial data, biological data, astrophysical data, and security logs.

The main part of the thesis consists of the following papers and articles published in reviewed proceedings of international conferences or in international journals with impact factor. The author contributions are listed where relevant. If the contributions are not stated explicitly, the first author of the paper was the main contributor.

Martin Kruliš, Tomáš Skopal, Jakub Lokoč, Christian Beecks: *Combining CPU and GPU Architectures for Fast Similarity Search*, Distributed and Parallel Databases, 30(3-4):179-207, 2012

David Bednárek, Michal Brabec, Martin Kruliš: *Improving Matrix-based Dynamic Programming on Massively Parallel Accelerators*, Information Systems, 64:175-193, 2017
*Author Contributions:* Martin Kruliš [60%] (general algorithm, GPU-related part); David Bednárek and Michal Brabec [40%] (Xeon Phi and CPU-related part)

Martin Kruliš, Steffen Kirchhoff, Jakub Yaghob: *Perils of Combining Parallel Distance Computations with Metric and Ptolemaic Indexing in kNN Queries*, In International Conference on Similarity Search and Applications, pages 127-138. Springer, 2014

Martin Kruliš, Hasmik Osipyan, Stéphane Marchand-Maillet: *Optimizing Sorting and Top-k Selection Steps in Permutation Based Indexing on GPUs*, In East European Conference on Advances in Databases and Information Systems, pages 305-317. Springer, 2015

Martin Kruliš, Hasmik Osipyan, Stéphane Marchand-Maillet: *Employing GPU Architectures for Permutation-based Indexing*, Multimedia Tools and Applications, 76(9):11859-11887, 2017

Martin Kruliš, Jakub Lokoč, Tomáš Skopal: *Efficient Extraction of Clustering-based Feature Signatures Using GPU Architectures*, Multimedia Tools and Applications, 75(13):8071-8103, 2016

David Honzátko, Martin Kruliš: *Accelerating Block-matching and 3D Filtering Method for Image Denoising on GPUs*, Journal of Real-Time Image Processing, pages 1-15, 2017
*Author Contributions:* David Honzátko [60%] (image denoising expertise, implementation); Martin Kruliš [40%] (CUDA expertise, supervision)

# CHAPTER 2

## Addressing Issues of Multimedia Databases

Multimedia databases such as photo galleries or video collections have to deal with slightly different problems than relational databases or other NoSQL databases [40, 50]. Perhaps the most important difference is their attitude towards searching and retrieval. Classical text-based searching in multimedia requires annotations of objects by tags or keywords which are difficult to obtain and may be inaccurate. This approach still prevails in some situations – e.g., when the multimedia are gathered from web pages which also provide additional content that can be used for text search. However, in multimedia databases where the content is generated by the users in large quantities, annotations has to be provided by the users who often neglect to do so.

Another approach is to search data by the analysis of their contents which is called *content-based retrieval* [24]. The objective is to extract semantic information from the database objects contents and utilize it in the search process. For instance, it may extract *keywords* or *concepts* (i.e., some form of annotations) which can be used for text-based search. It may also be used to introduce more complex types of queries like *query by example* or *query by sketch* [27]. The user is expected to provide a sample database object (e.g., an image) or an object abstraction (e.g., a shape sketched by a mouse or a digital pen). The system subsequently performs a *similarity search* where the contents of the query is compared to the contents of each database object and the objects that resemble the query the most are retrieved.

It may be difficult for the user to provide appropriate sample or to sketch decent query. It might be also difficult to retrieve desired objects at the first attempt as the similarity term may be subjective and less predictable by the users. Therefore, we can iterate the process by selecting an object from a result set and using it as a query in subsequent search. This exploration of the dataset [11] allows the user to refine the results with each iteration and increase the probability to retrieve the desire data.

In general, similarity search can be described as retrieving objects which are similar to given query. The similarity model typically comprise two things – *object descriptors*

and *distance function*. The descriptors typically extract important information from the objects so it can be accessed much faster or indexed. The distance function compares two descriptors and yields their dissimilarity – i.e., the lower the distance the more similar the objects are. Models may use simple signatures like vectors and Euclidean distance to compare them, or choose more complex signatures like *color histograms*, *MPEG-7* descriptors [33], *Scale-invariant Feature Transform* (SIFT) descriptors [32], or *Position-Color-Texture* (PCT) feature signatures [44]. These signatures also require more complex distance functions like the *Jaccard distance*, *Levenshtein's edit distance* [30], *Hausdorff metric* [28], *Signature Quadratic Form Distance* (SQFD) [12], or *Earth Mover's Distance* (EMD) [45].

A similarity search expects to retrieve the most similar objects to given query. We can limit either the number of retrieved objects (i.e., perform a *top-k* a.k.a. *kNN* query which yields $k$ the most relevant objects) or set a similarity threshold (i.e., a *range* query which yields all objects with distance lower than given constant $r$). The top-$k$ queries are the most typical ones as the user expects to see certain number of results on the screen.

When searching large datasets, the query evaluation process may be time demanding even with parallel processing. Therefore, an indexing method is required to narrow down the number of object candidates without computing the similarity function. Simple feature vectors may be represented as points multidimensional space $\mathbb{R}^d$ and traditional spatial indices may be applied for nearest neighbour search [25, 10].

More complex similarity models cannot be easily mapped to a vector space; however, we can at least make some assumptions about the distance function. In most cases, the distance function conforms to metric axioms, so *metric indexing* methods [14, 17, 41, 35] may be used to accelerate search queries. These methods use triangular inequality axiom to compute lower bound distance estimates for database objects from already computed distances between objects and query and from precomputed distances between selected database objects. If the lower bound estimate of an object exceeds the distances of objects already found in the intermediate top-$k$ result (or exceeding given range $r$), this object may be ruled out from the result immediately without actually computing the (possibly time demanding) distance function.

If the accuracy of the retrieved objects is less important than the processing speed, approximative methods may be used. Approximative similarity search [43] is a rather broad topic. It typically employs either reduction of dimensionality (mapping to a low-dimensional space) or reduction of comparisons (by aggressive pruning or early stopping) in order to reduce query processing time. We have focused on approximate indexing methods that are based on identifying nearest-neighbours of objects.

The most time-critical task is the query evaluation which involves distance computations and application of indexing methods. However, other tasks, such as construction of indices, similarity descriptors extraction, or data preprocessing might be important as well since large multimedia repositories have to process huge amounts of data uploaded by the users. We will address the performance issues of these tasks individually in the following sections.

## 2.1 Distance Functions for Similarity Search

In our research, we have focused on accelerating two similarity measures by the means of modern GPUs – the Signature Quadratic Form Distance [12] and the Levenshtein's edit distance [30]. We have also combined the SQFD with Linear Approximating Eliminating Search Algorithm (LAESA) – a metric indexing method [35] which improved the performance further and demonstrated that GPU accelerated distance computations can be combined with traditional indexing approach.

### 2.1.1 Accelerating Signature Quadratic Form Distance

Signature Quadratic Form Distance (SQFD) is a perfect candidate for parallelization. It is an adaptive measure which can compare signatures of different sizes yet it has regular structure. It is almost as effective as Earth Mover's distance (EMD), but it requires only $\mathcal{O}(N^2)$ time instead of $\mathcal{O}(N^4)$ time with respect to signature sizes. Furthermore, it is much more computationally demanding than simple Euclidean distance, so it will benefit from GPU acceleration.

The SQFD is a function that compares two *feature signatures* $S^q$ and $S^o$ representing the query and the object respectively. Signature is a set of features $S = \{(c_i, w_i)|i = 1 \dots n\}$, where each feature is a point in feature space $c_i \in \mathbb{R}^d$ and weight $w_i \in \mathbb{R}^+$. For the images, we used the Position-Color-Texture (PCT) feature space [44] which has 7 dimensions (2 for position, 3 for color in Lab space, 2 for texture properties).

The distance is formalized as

$$d_{SQFD_{f_s}}(S^q, S^o) = \sqrt{(w_q| - w_o) \cdot A_{f_s} \cdot (w_q| - w_o)^T}.$$

The vector $(w_q| - w_o)$ is created by concatenation of weight vectors $w_q$ and $-w_o$, where $-w_o$ has negated values. The concatenation of $w_q = (w_1^q, \dots, w_n^q)$ and $w_o = (w_1^o, \dots, w_m^o)$ looks like $(w_q| - w_o) = (w_1^q, \dots, w_n^q, -w_1^o, \dots, -w_m^o)$. The values $n$ and $m$ are shorthand notations of the sizes $|S^q|$ and $|S^o|$ respectively.

The similarity matrix $A_{f_s} \in \mathbb{R}^{(n+m)\times(n+m)}$ is the enumeration of similarity function $f_s$ applied to all pairs of centroids concatenated from both signatures. Let us denote $c = (c_q|c_o)$ the concatenation of $c_q$ centroids and $c_o$ centroids $(c_1^q, \dots, c_n^q, c_1^o, \dots, c_m^o)$. The elements of similarity matrix $A_{f_s}$ are then defined as $a_{ij} = f_s(c_i, c_j)$, where $i, j = 1, \dots, n + m$. Since the matrix represents all pairs of the concatenated centroid vector $c$. Function $f_s(c_i, c_j) \mapsto \mathbb{R}$ is the ground distance function that measures similarity between two centroids. In our work, we have exclusively used the Gaussian function with $L_2$ Euclidean metric – i.e., $f_s(c_i, c_j) = e^{-\alpha \cdot L_2^2(c_i, c_j)}$. The parameter $\alpha$ is a tuning parameter that affects indexability and precision of the model.

**Contributions**

Our work presented the first implementation of SQFD distance on GPUs. Furthermore, we have successfully combined parallel distance computations with metric indexing. This innovation opened possibilities for employing adaptive measures for real-time similarity search on contemporary image datasets utilizing reasonably priced hardware.

The greatest challenge of computing adaptive similarity measures on GPUs was fitting the workload on the hardware resources since the size of their inputs (i.e., the feature signatures) and intermediate values (the similarity matrix in case of SQFD) significantly exceeds the storage capabilities of a single GPU core. Therefore, our approach to SQFD parallelization is based on fine-grained decomposition where one distance is computed by a thread block and the threads cooperate in the matrix enumeration and subsequent reduction. The matrix and vector multiplications $(w_q| - w_o) \cdot A_{f_s} \cdot (w_q| - w_o)^T$ have been fused together with the $f_s(c_i, c_j)$ enumeration which provides maximal potential for parallelization. Furthermore, the input data as well as the data exchange performed in the final reduction utilize shared memory of the GPU streaming multiprocessors so the GPU cores are not stalled with global memory transfers.

The GPU adaptation of SQFD was combined with metric indexing methods to filter out candidate objects to which the distances are actually computed. This pre-filtering raises another issue of host-device data transfers as the feature signatures used for distance computations are no longer in one compact block. We employed multiple CPU threads to perform the metric index pre-filtering along with the gather operations to feed GPU steadily without stalls.

The experiments confirmed that the hybrid CPU-GPU solution that employed gaming GPUs NVIDIA GTX 580 outperformed a high-end 48-core NUMA server by an order of magnitude and serial baseline implementation by two orders of magnitude. Furthermore, when two GPUs were used instead of one, the solution scaled almost linearly. Details of our contribution are described in Chapter 3.

### 2.1.2 Accelerating Levenshtein's Edit Distance

Levenshtein's edit distance is a well established distance for comparing sequences such as text strings. The distance is defined as the minimal amount of given editing operations (like inserting, deleting, or replacing a character) to rewrite one sequence to another. Even though edit distance is often placed in relation with text similarity, it may be used for other sequences as well, such as time series or protein structures. We have selected this distance since it is quite time demanding yet very different from SQFD presented in the previous section. Furthermore, edit distance is a representative of a class of problems solved by dynamic computing.

Formally, the distance function is defined as

$$L_{a,b}(i,j) = \min(L_{a,b}(i-1,j), L_{a,b}(i,j-1), L_{a,b}(i-1,j-1) + \delta_{b_j}^{a_i}),$$

where the Kronecker $\delta$ compares the $i$-th and $j$-th positions in the input sequences $a$ and $b$ respectively. Additionally, we define borderline conditions: $L_{a,b}(i,0) = i$ and $L_{a,b}(0,j) = j$. The actual distance of sequences $a$ and $b$ is equal to $L_{a,b}(|a|,|b|)$.

Multiple algorithms to compute the distance exist. Perhaps the most direct approach would be to simply compute the distance recursively from its definition; however, such approach would lead to exponential algorithm. We can employ dynamic programming technique to avoid repetitive computations of the same value which is the main idea of the Wagner–Fischer algorithm [53]. The distance computation may be optimized further. For instance, we may observe that the adjacent values in the matrix constructed in the dynamic programming approach differs at most by one. Subsequent data compression combined with bit-vectorization leads to the Myers' algorithm [39]. On the other hand, the Wagner–Fisher represents an approach which can bee seen in many other algorithms like the Smith–Waterman algorithm [49] or the Dynamic Time Warping algorithm [38], so it has been selected as basis for our parallel implementation.

Internal parallelization of the edit distance is more complicated than of SQFD since the formula applied on every matrix element has inherent data dependency. Closer examination reveals [19] that this dependency still allows parallel processing of matrix elements lying on diagonals, but explicit synchronization is required between every diagonal. This synchronization needs to have very small overhead; otherwise, the benefits of parallel processing of the operations on diagonal are nullified. Furthermore, a parallel algorithm needs to deal with the fact that subsequent diagonals have different lengths.

Another challenge lies in data layout and data exchange between processing elements. Traditional optimization of serial Wagner–Fisher algorithm keeps only one row of the matrix in the memory during the computation. In case of diagonal approach, at least two preceeding diagonals to the one currently being computed has to be kept in memory. Furthermore, synchronizing data via main memory (i.e., global memory on GPUs) may be suboptimal, as the data exchange pattern between processing elements is very localized.

### Contributions

A GPU-accelerated version of Wagner–Fisher algorithm has been already proposed by Tomiyama et al. [52]. It utilizes the diagonal approach to parallelization and decompose the matrix into parallelogram-shaped stripes which provide more regular workloads. We have extended this method and proposed a generic SIMD algorithm which brings two important innovations. First, the decomposition is more flexible and recursive so it can be better adapted for various types of parallel hardware. Second, the algorithm keeps most of the intermediate data in registers and employ sophisticated shuffling operations[1] to ensure appropriate data exchange among processing elements. In addition, we optimized both single-distance and multi-distance versions of the algorithm – i.e., when only one distance on large inputs or many distances on smaller inputs are being computed in parallel.

[1] Warp-shuffle instructions in case of GPUs and SSE/AVX shuffle instructions in case of CPUs.

Proposed algorithm has been implemented and measured on mainstream multi-core CPU (with SSE instructions), Intel Xeon Phi (with AVX-512 instructions) and two NVIDIA GPUs – K20m and GTX 980. The measurements also helped us find optimal stripe dimensions for various input lengths (both for single-distance and multi-distance versions) and for each platform. Intel Xeon Phi as well as tested GPUs outperformed serial baseline by two orders of magnitude. Unfortunately, direct comparison with the previous work of Tomiyama was not possible but we have determined that the proposed optimizations increased performance up to 3× depending on the architecture and input sizes. The contribution is described in more detail in Chapter 4.

## 2.2 Parallel Processing and Metric Indexing

Metric indexing relies on a metric axiom of triangular inequality[2]. It states, that for any three different objects $\forall x \neq y \neq z$, the sum of distances $d(x, y) + d(y, z)$ must be greater or equal to $d(x, z)$. This inequality helps us to quickly estimate lower bounds for distances from already computed distances. Having two database objects $o_1, o_2$ and a query $q$ and having distance $d(q, o_1)$ already computed and distance $d(o_1, o_2)$ precomputed in the index, a *lower bound* of the distance is expressed as $d(q, o_2) \geq d(q, o_1) - d(o_1, o_2)$.

The lower bounds may be used to rule out data objects immediately without actually computing their (potentially expensive) distances. If a lower bound of a distance to an object is greater than current filtering range $r$, the object will never be included into the result set. In case of range queries, we have the $r$ value set as a constant parameter of the search. In case of top-$k$ queries, the $r$ is equal to the greatest distance in the intermediate top-$k$ result.

A metric index may be designed in various ways. For instance, the data may be clustered [14] and each cluster may be represented by one centroid object and radius (maximal distance to all objects in the cluster). If the distance to the centroid minus cluster radius is greater than the filtering range, we can prune all objects of the cluster from the search. Similarly, we can organize these clusters into M-tree [17] and test the lower bound estimates on every node to determine, whether the objects in a subtree are worth our attention.

For the purposes of parallel processing, we have selected another approach called Linear Approximating Eliminating Search Algorithm (LAESA) [35]. The algorithm is based on selecting fixed number of representative objects from the database called *pivots* or *vantage points*. The index itself is basically a table of precomputed distances between all database objects and all pivots (also denoted *pivot table*). At the beginning of a search, a distance between query object and each pivot is computed. Therefore, we can compute a lower bound estimate for every object using these distances and precomputed distances in pivot table simply by subtracting two numbers. Multiple pivots improve the lower bound estimate since the maximum of all pivot estimates is taken. Even though this type

[2]Actually, in our work we also worked with Ptolemy's inequality which provides better estimates. However, when focusing on the parallel aspects, the difference itself is not important and we have decided to omit the explanation for the sake of brevity.

of indexing may be computationally more demanding than list of clusters or M-tree, it may provide better selectivity and the lower bounds may be computed concurrently.

The greatest challenge is to combine LAESA indexing (i.e., prefiltering candidates) and parallel computation of distances in top-$k$ queries. The problem lies in updates of the intermediate top-$k$ result which also lowers the filtering range $r$. In sequential processing, the intermediate result may be updated with every distance computed and lower filtering range leads to a higher pruning rate, thus better effectivity of the index. On the other hand, we need to compute as many distances in parallel as possible to fully utilize GPU potential and reduce overhead.

Naïve approach is to compute distances in blocks and refine the intermediate result after every block. It has been used in our initial work in combining SQFD with metric indexing (Section 2.1.1) and it has proven adequate. However, further research revealed it is suboptimal both in the perspective of the parallel processing as the GPU stays idle for a nontrivial time between two subsequent blocks and the index effectiveness as the filtering range is lowered less often (i.e., more distances have to be computed).

#### Contributions

We have studied this problem thoroughly and found that the performance of the naïve approach depends heavily on selecting the appropriate block size for the given similarity model. An extensive empirical evaluation has determined optimal block sizes and pipeline setups for different distance function computational costs, number of pivots, and types of indexing (i.e., using triangular inequality as well as Ptolemy's inequality). Furthermore, we have derived simple patterns from the empirical data which provided guidelines to block size selection even without explicit measurements.

Naïve approach achieve significantly better performance in comparison with the full database scan conducted on GPU (ideal for parallelism) or the sequential implementation of LAESA index (which has highest possible effectivity). However, even with optimal block sizes it still remains suboptimal. Hence, we have devised a range estimation algorithm which basically converts top-$k$ queries into range queries (i.e., the filtering range is constant during the search). The estimation of optimal range is based on several heuristics and observations. In the fist stage, it computes distances for a carefully selected subset of the database. From these distances a range estimate is computed which is used to filter the rest of the database. This algorithm exhibits performance results which are much closer to theoretical performance optimum than the naïve approach and improve performance up to 2× with respect to naïve approach. The contribution is described in Chapter 5.

## 2.3 Approximative Indexing

In case of time demanding distance functions, searching the database may be rather slow even with metric indexing [46]. Furthermore, there are many tasks where the speed is prefered over accuracy, especially considering that the object similarity may be subjective.

For instance, when the user searches for similar images in a photo gallery, retrieving the results speedily is more important than returning the most accurate nearest neighbours as long as the results are still relevant. In such cases, approximative similarity search may be a better alternative [16, 29].

Permutation-based indexing (PBI) [9, 22, 42] is one of the most popular approximative techniques. The only constraint required by PBI is that the distance function conforms to the metric axioms. Similarly to other metric indexing methods (especially LAESA [35]), it selects a small subset of database objects called *pivots* or *vantage points*. Unlike LAESA, it keeps an ordered list of nearest pivots for each database object instead of the actual distances.

When the index is constructed, a permutation list $L(o_i, P)$ is computed for every object $o_i$. It comprise ordered indices of pivots so that the index of the nearest pivot is the first one on the list and if a pivot $p_i$ is before pivot $p_j$ on the list, $p_i$ is closer than $p_j$ to the corresponding database object. In many cases, storing the whole list is not necessary as the closest pivots are the most important ones. Therefore, a constant $\tilde{n} \leq |P|$ is selected and only indices of $\tilde{n}$ nearest pivots are stored.

The similarity of two objects (typically a database object and a query) is subsequently defined as the similarity of their permutation lists. Permutation lists can be compared by various functions – for instance, by Spearman's Rho measure [9] or Spearman Footrule Distance [36]. There are also many data structures that can handle permutation lists [8, 20, 36] and answer the queries. However, we have focused on the index construction which is a much more time-demanding task.

### Contributions

We have designed a hybrid CPU-GPU algorithm that computes the distances and constructs the permutation lists. Our solution significantly outperforms existing CPU-GPU solution by Mohamed et al. [37]. It provides valuable insights in optimizations of data-intensive algorithms such as computing Euclidean distances between two groups of vectors in high-dimensional space. We have also devised a novel approach to parallel sorting and top-$k$ selections of smaller sets being computed concurrently.

First, we focused on computing the list of nearest $\tilde{n}$ pivots for each object once we know the distances. Basically, it requires a combination of top-$k$ selection and traditional sorting. The list is constructed for every database object, so multiple lists may be constructed concurrently; however, assigning one list per GPU thread is not possible due to limited resources. Furthermore, both top-$k$ selection algorithms (such as maintaining 2-regular heap) and most sorting algorithms are data-driven and unstable – thus not very suitable for lock-step execution.

We have selected an approach where one list is constructed by one thread block. Threads in a block execute cooperatively a modified version of bitonic sorting, which is used both for selecting nearest $\tilde{n}$ pivots as well as for sorting the list. Performing this task on a GPU improves the speed of the list construction up to a factor of 5 (depending on the $\tilde{n}$). More

importantly, it reduces the amount of data required to be transferred from GPU to CPU since only $\tilde{n}$ values (instead of $|P|$) are kept per object and the actual distances are no longer relevant for the index. This index construction step was presented separately in Chapter 6 so we can discuss it in more detail.

Subsequently, we used the modified bitonic sorting algorithm for the construction of permutation-based index of high-dimensional Euclidean spaces. The Euclidean metric is very straightforward to implement; however, it requires only a few computing operations for every input data element which makes it very sensitive to data layout patterns and caching. Furthermore, both database object vectors and pivot vectors are used in multiple distance computations so a smart work organization may improve the balance between data transfers and computations.

We have tested multiple approaches to data organization, caching (including manual caching in shared memory), and work distribution among the threads. Implemented prototypes were measured on NVIDIA GTX 980 as well as K20m and fastest solutions were selected. The solutions were combined with previously mentioned algorithm for top-$k$ selection and sorting. The whole index construction was accelerated 10× on a NUMA server with K20m GPUs and 46× on a desktop PC with GTX 980 GPUs. It is worth mentioning that the implemented solution reached the I/O limits of the persistent storage which comprised six SSD drives in RAID0 configuration. Detailed description is presented in Chapter 7.

## 2.4 Extracting Feature Signatures on GPU

The feature signature extraction is essential for similarity models. However, signatures are typically extracted from each database object only once. One of the reasons is that the extraction process is time demanding and nobody wishes to spent hardware resources to repeatedly compute descriptors which may be cached easily. On the other hand, accelerating the extraction process will not only improve efficiency, but also open new possibilities like adaptive reindexing – i.e., allowing the signatures to be re-computed with different tuning parameters for the similarity model to adjust the effectivity of the similarity search dynamically.

We have been operating with position-color-texture (PCT) signatures [44], which were also used in the experiments with SQFD acceleration on GPUs (Section 2.1.1). A signature $S$ is a set of features, which are points from feature space $F_s$. In our case, the feature space is in fact a subset of $\mathbb{R}^7$ interpreted as $f = (x, y, L, a, b, c, e)$. The $(x, y)$ coordinates represent a position within the image, $(L, a, b)$ properties hold the color information (represented in Lab color space [34]), and $(c, e)$ stands for contrast and entropy values respectively which characterize texture properties [23].

Each feature $f$ from the signature $S$ is accompanied by a weight $w_f \in \mathbb{R}^+$, that summarizes its importance (i.e., how lare area of the image the feature represents). This way, a larger area with the same color and texture can be represented by a single feature point with

greater weight while a more heterogeneous area is represented by multiple feature points with smaller weights.

The extraction algorithm has two parts. First, the image is sampled and features are extracted using a large number of random coordinates $(x, y)$. Subsequently, the features are aggregated using modified k-means clustering algorithm [26]. The k-means produces cluster centroids which are taken as final features for the signature and cluster sizes which become feature weights.

The extraction of color features is straightforward, since it only requires transforming the color of a pixel from RGB to Lab color space. Computing the texture features is far more complicated. First of all, the image is transformed into coarse greyscale (e.g., 4-bits per pixel). Vicinity of each feature point (represented by a rectangular window of fixed size) is examined and *co-occurrence* matrix $\Gamma$ is constructed. Each matrix value $\gamma_{i,j}$ holds a number of total adjacent pixel pairs[3] within the window where one pixel has value $i$ and the other one has value $j$ (on the coarse greyscale). Given $G_s$ the number of shades of grey and $\bar{\gamma}$ normalized values of matrix $\Gamma$, the contrast $c$ and entropy $e$ are computed as:

$$c = \sum_{i=0}^{G_s} \sum_{j=0}^{i} (i-j)^2 \cdot \bar{\gamma}_{i,j}, \quad e = \sum_{i=0}^{G_s} \sum_{j=0}^{i} -\ln(\bar{\gamma}_{i,j}) \cdot \bar{\gamma}_{i,j}$$

The k-means algorithm is an approximative method for determining the Voronoi partitioning of points in Euclidean space. It finds an explicit division of points into $k$ clusters so that each point belongs exactly to one cluster. The clusters are represented by centroids (means) which are selected randomly at the beginning. In each iteration, the algorithm determines assignment of points into clusters by finding their nearest means and computes new centroids as centers of mass of points assigned to their corresponding clusters. The algorithm was slightly modified to suit the needs of adaptive feature aggregation:

- The final signature requires only centroids and weights; hence, we do not need to construct the point assignments explicitly.
- The signature needs to be adaptive so we use fixed number of refining iterations instead of fixed number of clusters ($k$).
- To reduce noise, we dismiss small clusters and join clusters which have their means close together.

### Contributions

We have designed and implemented the first feature signature extractor accelerated by GPUs. To our best knowledge, this was the first attempt for extracting complex adaptive signatures from images using massively parallel architectures. The implementation itself not only contributed to our efforts of improving performance of multimedia database systems, but it also allowed us to conduct thorough experiments with various setting of

[3] Adjacent pixels share a side or a corner.

studied similarity models. These experiments were conducted in the matter of days even though they would have taken more than a year if executed sequentially on a CPU.

The greatest challenge was managing the data in memory in order to keep hot data in shared memory or memory caches. Our approach is to compute one image by one thread-block since a batch of feature signatures is typically being extracted simultaneously. We have devised a word-aligned format to cache greyscale image in shared memory and fit as many co-occurrence matrices as possible (given selected greyscale and window size). The matrix is also stored in a word-aligned compacted format and the computations run in parallel using a combination of atomic bitwise operations to ensure correct value increments.

Subsequent k-means clustering takes advantage of the fact that one image is being processed by one thread block, so the synchronization between individual steps of each iteration may be done by barriers. The assignment of points to clusters is not constructed explicitly, but the information is immediately used for the computation of means for the next iteration. Data synchronization was realized by clever application of atomic instructions.

The improvements have been evaluated empirically using four GPUs NVIDIA Tesla M2090. We have also conducted thorough study of the impact of various parameters (affecting both the feature extraction and clustering) on the performance. Our contribution is presented in Chapter 8. Similarly to the previous database problems we have addressed, the speedup exceeded of two orders of magnitude with respect to the serial version and the solutions scales almost linearly with available GPUs.

## 2.5 Similarity Approach to Image Denoising

Many public services for managing and publishing multimedia (like Instagram[4] or Youtube[5]) offer some form of semi-automated improvements of images or videos, or even means for editing the contents. Although such operations may stand on the edge of interest of multimedia databases in general, data quality still may be an issue for some users. Such tasks are not considered time-critical and does not always require real-time processing; however, like in the case of feature extraction, GPUs may lead to better efficiency which could reduce power consumption and hardware costs.

In case of photo galleries, one of the greatest issue is removing the image noise which is caused by imperfections of optical sensors in digital cameras used for acquiring the images. We have selected a method called *Block-matching and 3D Filtering* [18] which is considered very effective in the terms of the denoising performance, but its CPU implementation is rather slow. It peaked our interest for two reasons – it employs self similarity of image patches (which is very related to our other research interests) and its GPU acceleration may be directly employed outside the realm of databases to improve other use cases (such as processing photographs in a graphical editor) so it may have greater impact. The

[4] https://www.instagram.com/
[5] https://www.youtube.com/

method is rather complex so we focus mainly on the details which are important for the parallelization process.

As the name suggests, the method comprise two main steps. First, rectangular patches (blocks) of the denoised image are matched and grouped based on their similarity into 3D stacks[6]. After that, each stack is processed by a filtering function which performs hard thresholding on decorrelated data. The final result is aggregated from the denoised data of all patches.

The block-matching step is perhaps the most time-demanding. It requires that each patch of the input image is compared with all other patches. Therefore it has time complexity $\mathcal{O}(w^2h^2)$, where $w$ and $h$ denotes the *width* and the *height* of the image respectively. Simple Euclidean metric is used to compare the patches and a list of $k$ nearest patches is maintained for each patch of the image. Even though the existing implementations employ various simplifications that reduce the number of patches which are considered for comparison, the procedure may easily take up several minutes on contemporary CPUs when denoising regular photographs.

The second step gathers the similar patches matched in the first step, compose 3D stacks from them, and perform the filtering. Each 3D stack is transformed by a decorrelating function (e.g., Fourier, Cosine, Bior, or Walsh-Hadamard transform). The transformed coefficients are subjected to hard thresholding which means that values lower than given threshold are zeroed. Finally, the coefficients are transformed back using corresponding inverse transformation. Since the patches may overlap and one patch may be present in multiple stacks, the denoised data are gathered from all the patches and averaged at the end.

### Contributions

We have designed and implemented a novel CUDA algorithm for BM3D method which significantly outperforms the only existing GPU implementation in OpenCL [48]. Our work also provides a complete solution – i.e., it supports two phase extension where the first run (producing basic estimate) is followed by a second run employing Wiener filtering. Furthermore, the implementation is heavily parametrized and may be even used for testing other types of decorrelating transformations.

The BM3D method itself provides many challenges for optimization and parallelization. Typical photographs have sizes in megapixels, so it might be easy to accomodate multiple images in GPU global memory, but only a small part of one photograph may fit caches or shared memory. Furthermore, the block-matching algorithm operates in quadratic time with respect to image size and could require much more memory than the denoised image itself. To avoid these problems, we have designed the algorithm to process one image at a time and most of the steps are divided into smaller batches that process only part of the image. On the other hand, the method can be easily modified to process multiple images concurrently in case their sizes are much smaller than typical photographs.

[6]Multiple 2D patches of the same size are grouped in the third dimension creating a 3D stack.

The block-matching algorithm has been designed in a way that promotes thread cooperation, aggregates partial distance computations from overlapping patches, and makes a good use of shared memory for manual data caching. The updates of constructed similarity lists are designed to be lock-free, since each thread use privatized copy of related lists and these copies are aggregated at the end using implicit synchronization.

The collaborative filtering, which comprise the 3D transformations and hard thresholding, was aggregated into a single kernel so that each 3D stack of patches is constructed and kept only in the shared memory of the streaming processors. The filtered patches are stored back to aggregation buffers where the result image is being assembled by the means of atomic instructions.

The whole implementation was subjected to extensive empirical evaluation on three GPU platforms (NVIDIA Tesla K20m, GTX 980 and GTX 1080) and compared with previous CPU (OpenMP) implementation and OpenCL implementation. We have achieved speedup over 50× with respect to the parallel CPU baseline and over 10× when compared to referential OpenCL implementation. The results also include detailed analysis of individual algorithm steps and scaling properties of the algorithm when employed on different image sizes. The details are summarized in Chapter 9.

# CHAPTER 3

## Combining CPU and GPU Architectures for Fast Similarity Search

**Authors: Martin Kruliš, Tomáš Skopal, Jakub Lokoč, Christian Beecks**

[7] *Distributed and Parallel Databases, 30(3-4):179-207, 2012, DOI: 10.1007/s10619-012-7092-4*

# CHAPTER 4

## Improving Matrix-based Dynamic Programming on Massively Parallel Accelerators

**Authors: David Bednárek, Michal Brabec, Martin Kruliš**

[1] *Information Systems, 64:175-193, 2017*
*DOI: 10.1016/j.is.2016.06.001*

# CHAPTER 5

## Perils of Combining Parallel Distance Computations with Metric and Ptolemaic Indexing in kNN Queries

**Authors: Martin Kruliš, Steffen Kirchhoff, Jakub Yaghob**

[3] *In International Conference on Similarity Search and Applications, pages 127-138. Springer, 2014, DOI: 10.1007/978-3-319-11988-5_12*

# CHAPTER 6

## Optimizing Sorting and Top-k Selection Steps in Permutation Based Indexing on GPUs

**Authors: Martin Kruliš, Hasmik Osipyan, Stéphane Marchand-Maillet**

[5] *In East European Conference on Advances in Databases and Information Systems, pages 305-317. Springer, 2015, DOI: 0.1007/978-3-319-23201-0_33*

# CHAPTER 7

## Employing GPU Architectures for Permutation-based Indexing

**Authors: Martin Kruliš, Hasmik Osipyan, Stéphane Marchand-Maillet**

[6] *Multimedia Tools and Applications, 76(9):11859-11887, 2017*
*DOI: 10.1007/s11042-016-3677-7*

# CHAPTER 8

## Efficient Extraction of Clustering-based Feature Signatures Using GPU Architectures

**Authors: Martin Kruliš, Jakub Lokoč, Tomáš Skopal**

[4] *Multimedia Tools and Applications, 75(13):8071-8103, 2016*
*DOI: 10.1007/s11042-015-2726-y*

# CHAPTER 9

## Accelerating Block-matching and 3D Filtering Method for Image Denoising on GPUs

**Authors: David Honzátko, Martin Kruliš**

[2] *Journal of Real-Time Image Processing, pages 1-15, 2017*
*DOI: 10.1007/s11554-017-0737-9*

# CHAPTER 10

# Conclusion and Future Work

This thesis provides an overview of selected issues from the domain of multimedia databases. Our main objective was to accelerate relevant algorithms and methods by employing parallel processing, GPU architectures in particular. The studied problems ranged from similarity search to image preprocessing. Together, studied issues cover all important aspects of multimedia databases and presented contributions provide valuable insights in GPU algorithm design which may be also beneficial beyond the domain of databases.

In our efforts to accelerate content-based retrieval methods, two similarity distance functions were selected as representatives and candidates for parallelization. The Signature Quadratic Form Distance is a rather complex function designed to compare adaptive feature signatures. The Levenshtein's Edit Distance is a well known measure for comparing sequences and its traditional implementation (the Wagner–Fischer algorithm) employs dynamic programming approach which is typical for some other problems as well. In both cases, the algorithms computing the distances had to be modified significantly to accomodate nuances of GPU architectures, especially the lock-step code execution and limited memory resources. In case of Wagner–Fischer algorithm, we have extended our interest beyond the domain of GPUs and designed a more universal SIMD solution which was implemented for modern CPUs and Xeon Phi devices as well.

Even when accelerated by GPUs, computing distances between a query and every database object is not very efficient. Therefore, parallel processing needs to be combined with indexing methods. Unfortunately, most forms of indexing require to know a filtering range which is a threshold for pruning dissimilar objects. In case of $k$NN queries, this range is decreasing with every refinement of intermediate top-$k$ set – i.e., possibly with every distance computed. Our parallel algorithms for distance computations needs to operate in batches to fully utilize the GPU with reasonable overhead which prevents frequent updates of the intermediate top-$k$ result, thus reducing the effectivity of indexing. We have addressed this problem and designed a range estimation method which eliminates the problem of frequent updates and allow the query evaluation process to fully exploit parallelism whilst maintaining a high level of index effectiveness.

Besides the conservative ways of evaluation of content-based queries, we also explored approximative approach. In many use cases the user does not actually require the most accurate results, especially since the similarity is often a subjective term. Providing adequate results in more speedily fashion could be prefered in such cases. We have selected Permuation-based Indexing method as a representant of approximative methods. Its utilization for searching is rather straightforward, so we focused on its construction when indexing high-dimensional Euclidean spaces. The accelerated version of the index had to solve several issues which are rather interesting even on their own – especially the efficient computation of multiple $L_2$ distances in high-dimensional space (which involves elaborate data organization and caching), selecting $k$ nearest neighbours for each pivot in parallel, and sorting these neighbours indices by their distance on the GPU.

The content-based retrieval operates with object descriptors which need to be extracted from all database object as well as from the query. This extraction is a part of the data preprocessing which usually takes place only when objects are being inserted or updated. On the other hand, most types of descriptors extraction is rather time demanding and it would be beneficial to accelerate this task as well since we already assume the database system has GPUs at its disposal. We have developed a GPU-accelerated extractor for image Position-Color-Texture feature signatures, which can be used along with the SQFD function. The most challenging part was the extraction of texture features based on co-occurrence matrices. We have designed a parallel algorithm which employs bitwise data compression along with atomic instructions for updates. The features are subsequently compacted into signatures by modified k-means clustering. Despite the fact this algorithm is well known, we had to redesign it for the GPU as the modifications needed to be fused inside and the implementation had to be tailored specifically for the signatures to achieve optimal performance.

Last topic addressed by this thesis is image denoising – i.e., removing noise from physically acquired images such as photographs. Although this task is only loosely related to databases, improving data quality may be important for some systems such as photo galleries. Furthermore, we have selected Block-matching 3D Filtering method which exploits image self-similarity to improve traditional filtering techniques. The method is also rather time demanding, so it could really benefit from fast parallel processing. We have designed and implemented the BM3D algorithm for GPUs in a way, so it can be used on a single image (e.g., in a photo editor) or simply employed for batch processing in database systems.

Our proposed algorithms were subjected to extensive empirical evaluation. In all cases, we have observed a speedup over two orders of magnitude with respect to the serial CPU baseline and over an order of magnitude with respect to the parallel CPU implementation. Such speedup is to be expected when comparing raw computational power of the two architectures, so it is our belief that presented solutions are very near their theoretical optimums.

### Future Work

Despite the practical nature of the presented research, it still lies firmly in the realms of basic research. Although the presented methods contributed to the research in the domain of multimedia, content-based retrieval, and GPU-accelerated algorithms (as suggested by paper citations), we did not provide any data from large-scale applications. To advance this work further, it has to be employed in real multimedia databases and measured under actual workloads. However, such advance would most definitely require participation of a commercial subject already involved in multimedia databases.

Contemplating our choice regarding the parallel hardware platform, we must conclude that the GPUs are still probably the best option since they have become mainstream accelerators for various tasks beyond mere computer graphics. On the other hand, it might be interesting to expand the studied methods to other hardware platforms as well. Field-programmable Gate Arrays (FPGAs) might be our next choice for consideration as they are becoming more and more popular recently. FPGAs could improve power-efficiency of most algorithms and even provide faster solutions than GPUs in some cases. Furthermore, their programming model is rather specific and complex so providing easier ways how to introduce them into database systems will also require additional research.

Finally, we have observed that all our parallel algorithms exhibited certain similarities. Even though the GPU architectures are very complex and it require extensive training to design optimal parallel code for lock-step execution, it might be possible to achieve some level of code optimization or code design using modern methods of machine learning, especially deep learning with neural networks. If we are able to train models for automated GPU code optimization it would save a lot of work and lot of research as well.

# Bibliography

## Included Publications


[1] D. Bednárek, M. Brabec, and M. Kruliš. Improving matrix-based dynamic programming on massively parallel accelerators. *Information Systems*, 64:175–193, 2017.

[2] D. Honzátko and M. Kruliš. Accelerating block-matching and 3d filtering method for image denoising on gpus. *Journal of Real-Time Image Processing*, pages 1–15, 2017.

[3] M. Kruliš, S. Kirchhoff, and J. Yaghob. Perils of combining parallel distance computations with metric and ptolemaic indexing in knn queries. In *International Conference on Similarity Search and Applications*, pages 127–138. Springer, 2014.

[4] M. Kruliš, J. Lokoč, and T. Skopal. Efficient extraction of clustering-based feature signatures using gpu architectures. *Multimedia Tools and Applications*, 75(13):8071–8103, 2016.

[5] M. Kruliš, H. Osipyan, and S. Marchand-Maillet. Optimizing sorting and top-k selection steps in permutation based indexing on gpus. In *East European Conference on Advances in Databases and Information Systems*, pages 305–317. Springer, 2015.

[6] M. Kruliš, H. Osipyan, and S. Marchand-Maillet. Employing gpu architectures for permutation-based indexing. *Multimedia Tools and Applications*, 76(9):11859–11887, 2017.

[7] M. Kruliš, T. Skopal, J. Lokoč, and C. Beecks. Combining cpu and gpu architectures for fast similarity search. *Distributed and Parallel Databases*, 30(3-4):179–207, 2012.


## Referenced Publications


[8] G. Amato, C. Gennaro, and P. Savino. Mi-file: using inverted files for scalable approximate similarity search. *Multimedia tools and applications*, 71(3):1333–1362, 2014.

[9] G. Amato and P. Savino. Approximate similarity search in metric spaces using inverted files. In *Proceedings of the 3rd international conference on Scalable information systems*, page 28. ICST (Institute for Computer Sciences, Social-Informatics and ..., 2008.

[10] R. Bayer. The universal b-tree for multidimensional indexing: General concepts. In *International Conference on Worldwide Computing and Its Applications*, pages 198–209. Springer, 1997.

[11] C. Beecks, T. Skopal, K. Schöffmann, and T. Seidl. Towards large-scale multimedia exploration. In *Proceedings of the 5th International Workshop on Ranking in Databases (DBRank 2011)*, pages 31–33, 2011.

[12] C. Beecks, M. S. Uysal, and T. Seidl. Signature quadratic form distance. In *Proceedings of the ACM International Conference on Image and Video Retrieval*, pages 438–445. ACM, 2010.

[13] S. Bell and K. Bala. Learning visual similarity for product design with convolutional neural networks. *ACM Transactions on Graphics (TOG)*, 34(4):98, 2015.

[14] E. Chávez and G. Navarro. An effective clustering algorithm to index high dimensional metric spaces. In *String Processing and Information Retrieval, 2000. SPIRE 2000. Proceedings. Seventh International Symposium on*, pages 75–86. IEEE, 2000.

[15] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014.

[16] P. Ciaccia and M. Patella. Pac nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces. In *Proceedings of 16th International Conference on Data Engineering (Cat. No. 00CB37073)*, pages 244–255. IEEE, 2000.

[17] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proceedings of the Twenty-third International Conference on Very Large Data Bases (VLDB)*, pages 426–435. Morgan Kaufmann, 1997.

[18] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. Image denoising with block-matching and 3d filtering. In *Image Processing: Algorithms and Systems, Neural Networks, and Machine Learning*, volume 6064, page 606414. International Society for Optics and Photonics, 2006.

[19] G. Delgado and C. Aporntewan. Data dependency reduction in dynamic programming matrix. In *2011 Eighth International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pages 234–236. IEEE, 2011.

[20] A. Esuli. Mipai: Using the pp-index to build an efficient and scalable similarity search system. In *2009 Second International Workshop on Similarity Search and Applications*, pages 146–148. IEEE, 2009.

[21] D. Geer. Chip makers turn to multicore processors. *Computer*, 38(5):11–13, 2005.

[22] E. C. Gonzalez, K. Figueroa, and G. Navarro. Effective proximity retrieval by ordering permutations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(9):1647–1658, 2008.

[23] C. C. Gotlieb and H. E. Kreyszig. Texture descriptors based on co-occurrence matrices. *Computer Vision, Graphics, and Image Processing*, 51(1):70–86, 1990.

[24] V. N. Gudivada and V. V. Raghavan. Content based image retrieval systems. *Computer*, 28(9):18–22, 1995.

[25] A. Guttman. *R-trees: A dynamic index structure for spatial searching*, volume 14. ACM, 1984.

[26] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[27] K. Hirata and T. Kato. Query by visual example. In *international conference on extending database technology*, pages 56–71. Springer, 1992.

[28] D. P. Huttenlocher, G. A. Klanderman, and W. A. Rucklidge. Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (9):850–863, 1993.

[29] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

[30] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet physics doklady*, volume 10, page 707, 1966.

[31] N. Lopes and B. Ribeiro. Gpumlib: An efficient open-source gpu machine learning library. *International Journal of Computer Information Systems and Industrial Management Applications*, 3:355–362, 2011.

[32] D. G. Lowe et al. Object recognition from local scale-invariant features. In *iccv*, volume 99, pages 1150–1157, 1999.

[33] B. S. Manjunath, P. Salembier, and T. Sikora. *Introduction to MPEG-7: multimedia content description interface*. John Wiley & Sons, 2002.

[34] K. McLaren. XIII–The Development of the CIE 1976 (L* a* b*) Uniform Colour Space and Colour-difference Formula. *Journal of the Society of Dyers and Colourists*, 92(9):338–341, 1976.

[35] M. Micó, J. Oncina, and E. Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements. *Pattern Recognition Letters*, 15(1):9–17, 1994.

[36] H. Mohamed and S. Marchand-Maillet. Quantized ranking for permutation-based indexing. In *International Conference on Similarity Search and Applications*, pages 103–114. Springer, 2013.

[37] H. Mohamed, H. Osipyan, and S. Marchand-Maillet. Multi-core (cpu and gpu) for permutation-based indexing. In *International Conference on Similarity Search and Applications*, pages 277–288. Springer, 2014.

[38] M. Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007.

[39] G. Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *Journal of the ACM (JACM)*, 46(3):395–415, 1999.

[40] S. Nepal and M. Ramakrishna. Query processing issues in image (multimedia) databases. In *Proceedings 15th International Conference on Data Engineering (Cat. No. 99CB36337)*, pages 22–29. IEEE, 1999.

[41] D. Novak and M. Batko. Metric index: An efficient and scalable solution for similarity search. In *Similarity Search and Applications, 2009. SISAP'09. Second International Workshop on*, pages 65–73. IEEE, 2009.

[42] D. Novak, M. Kyselak, and P. Zezula. On locality-sensitive indexing in generic metric spaces. In *Proceedings of the Third International Conference on SImilarity Search and APplications*, pages 59–66. ACM, 2010.

[43] M. Patella and P. Ciaccia. Approximate similarity search: A multi-faceted problem. *Journal of Discrete Algorithms*, 7(1):36–48, 2009.

[44] Y. Rubner and C. Tomasi. *Perceptual metrics for image database navigation*, volume 594. Springer Science & Business Media, 2013.

[45] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99–121, 2000.

[46] H. Samet. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006.

[47] J. Sanders and E. Kandrot. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

[48] S. Sarjanoja, J. Boutellier, and J. Hannuksela. Bm3d image denoising using heterogeneous computing platforms. In *2015 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, pages 1–8. IEEE, 2015.

[49] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

[50] V. Subrahmanian and S. Jajodia. *Multimedia database systems: issues and research directions*. Springer Science & Business Media, 2012.

[51] T. N. Theis and H.-S. P. Wong. The end of moore's law: A new beginning for information technology. *Computing in Science & Engineering*, 19(2):41, 2017.

[52] A. Tomiyama and R. Suda. Automatic parameter optimization for edit distance algorithm on gpu. In *International Conference on High Performance Computing for Computational Science*, pages 420–434. Springer, 2012.

[53] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173, 1974.

[54] M. M. Waldrop. The chips are down for moore's law. *Nature News*, 530(7589):144, 2016.