Comparative analysis of exhaustive searching on a massive finger-vein database over multi-node/multi-core and multi-GPU platforms

Sebastián Guidet[1], Ruber Hernández-García[2], Fernando Emmanuel Frati[1], and Ricardo J. Barrientos[2,3]

[1] Department of Basic and Technological Sciences,
Universidad Nacional de Chilecito, Chilecito, La Rioja, Argentina
sguidet@undec.edu.ar, fefrati@undec.edu.ar
[2] Laboratory of Technological Research in Pattern Recognition (LITRP),
Faculty of Engineering Sciences, Universidad Católica del Maule, Talca, Chile
rbarrientos@ucm.cl, rhernandez@ucm.cl
[3] Department of Computer Science and Industries, Faculty of Engineering Science,
Universidad Católica del Maule, Talca, Chile

Abstract. When searching on unstructured data (video, images, etc.), response times are a critical factor. In this work we propose an implementation on two types of multi-GPU and multi-node/multi-core platforms, for massive searches. The presented method aims to reduce the time involved in the search process by solving simultaneous queries over the system and a database of millions of elements. The results show that the multi-GPU approach is 1.6 times superior to the multi-node/multi-core algorithm. Moreover, in both algorithms the speedup is directly proportional to the number of nodes reaching 156x for 4 GPUs, and 87x in the case of the hybrid multi-node/multi-core algorithm.

Keywords: High Performance Computing, identification of individuals, Local linear binary pattern, Finger veins, GPU.

1 Introduction

The volume of data in the world is growing exponentially. According to some estimates by the united nations, 90% of the world's data has been created in the last two years and is predicted to grow at 40% per year. In 2020, 64.2 zettabytes of data were created, a 314% increase over 2015 [7]. Most of this data is unstructured (videos, images, etc.), so by needing to perform a search it cannot be treated in the same way as traditional databases. In multimedia databases it is not possible to perform exact searches, because the information is not always stored in the same way. Two images may look the same, but when compared pixel by pixel they are totally different. In this type of databases, similarity search is used, which consists of retrieving all those objects within a database that are similar according to a given query.

Similarity searches are mathematically modeled through metric spaces using a distance function [1]. The distance function is responsible for determining the degree of similarity between two objects. As the size of the database increases, the system must perform more distance calculations. Consequently, the performance of similarity searches is largely conditioned by the distance function computation and the size of the database, which influences the total processing time.

Due to the computational load involved in the search process and its impact on the processing time, it is necessary to search for alternatives to speed up the computation involved in solving queries on the multimedia database. One of the most widely used methods in the literature to reduce costs in terms of time is through parallel processing. Although the parallelization of algorithms is not a new topic, the emergence of GPU coprocessors nowadays allows a high level of parallelism at a very low cost.

In this paper, we propose a comparison between multi-node/multi-core and multi-GPU parallel methods, applying an exhaustive search algorithm to solve k-NN queries in metric spaces. As a case study we use a database of finger vein images composed of 40,000,000 images, preprocessed by the CLAHE (limited contrast histogram adaptive histogram equalization) algorithm and the vertical LLBP (vertical linear local binary pattern) descriptor. In the experiments performed we used the Hamming distance to measure the similarity between two images.

2 Background on similarity search

The search for objects in a database that are similar to a given query object is a problem that has been extensively studied in recent years. The solutions are based on two criteria: the first one determines how the selection of the solution set will be performed when searching; the second one refers to the way in which the database is traversed to apply the first criterion.

For the selection of the solution set, the model of metric spaces [1] is used. A metric space (X, d) consists of a universe of valid objects X and a distance function $d : XxX \rightarrow R^+$ defined between them. The distance function determines the similarity between two given objects. The finite subset $U \subset X$ with size $n = |U|$, is called the database and represents the collection of objects in the search space. There are two main queries of interest, kNN and range queries.

Range query[1]: the objective is to retrieve all objects $u \in U$ within a radius r of the query q.

The k nearest neighbors (kNNN)[2]: the goal is to recover the set $kNNN(q) \subseteq U$ such that $|kNNN(q)| = k$ and $forallu \in kNN(q); v \in U - kNN(q); d(q, u) \leq d(q, v)$.

When working with large databases, even if the radius is small, range queries provide large solution sets [4]. Therefore, in these cases the most commonly used and efficient query method is k-nearest neighbor queries. However, when applying the k-nearest neighbors method, it is necessary to calculate the distance between the query and each element belonging to the database. For this calculation, the Hamming distance similarity function is used, whose effectiveness has been proven by other authors in the literature [8].

When performing the database traversal to resolve similarity queries in metric spaces, the most trivial but costly (in time and/or resources) method is the brute force method. An alternative is the index-based methods [3], but they have the disadvantage of losing efficiency as the size of the database increases, in addition to presenting problems in shared memory systems [2].

3 Searching process on a massive database

The identification of individuals consists of an exhaustive 1:N search in the database, this procedure returns a list of 32 records sorted by similarity score in ascending order. We only get the first 32 results because it is the lowest perfect recognition rank for $LLBPv$ with the best [6] precision performance.

This process must be performed for each query received by the system. Thus, the system workload increases with a high rate of queries per unit time, and the volume of data to be processed increases significantly, so the response time must be reduced.

3.1 Multi-node/multi-core searching algorithm

The presented approach attempts to reduce the computation time of 1:N similarity comparisons by achieving a significant speedup for an exhaustive search process on a massive database. For this, a hybrid multi-node/multi-core parallel version was implemented in order to distribute the tasks among the different nodes of the cluster. Therefore, each node computes the similarity tests on a partition of the database. Each node handles its tasks by applying the Round Robin ($dist - round - Robins$) distribution scheme [6]. Each query is solved with 8 threads, as each node has 36 available threads, each node of the algorithm can process up to 4 queries in parallel.

Once all the processes have completed their tasks in parallel, a single array is generated with the 32 elements of smallest distances in increasing order. Finally, each node sends its 32 shortest distance results (local results) to the master node, which computes the final 32 results.

3.2 Multi-GPU searching algorithm

In order to process a larger amount of items in the shortest possible time, a multi-GPU algorithm is implemented, where a multi-threaded session is started on the CPU to handle each GPU (one GPU per thread). The algorithm divides the database into parts, each part will be processed by a different GPU until the number of items in the database is completed. It should be noted that each GPU processes its portion of the database by implementing the algorithm described in [5], where each query is solved with a different CUDA block. Finally, the quicksort algorithm is used to sort and return the 32 smallest distances of the process. It should be noted that each GPU of the multi-GPU platform can only process 56 simultaneous queries due to the occupancy factor of the model used [5].

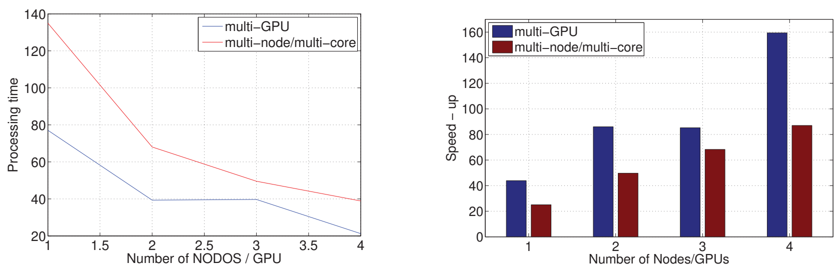
4 Experimental results

The experimental environment is composed of 4 servers (or nodes), and each node with 2 Intel Xeon Gold 6140 CPUs @ 2.30 GHz, totaling 36 physical cores, 24.75 MB of L3 cache and 126 GB of RAM. In addition, one of these servers has 4 GPU model NVIDIA GeForce GTX 1080 TI, CUDA Cores: 3584, GPU Memory: GDDR5X 11GB.

To evaluate the ability of the proposed algorithm to respond to simultaneous queries in adequate time, the BigFVDB dataset generated in previous work is used [6], using 4,000,000 samples for these experiments.

The experiments were performed by increasing the number of processing nodes in both architectures up to 4 nodes. To obtain an unbiased result and ensure the stability of the results, the time measurements were averaged by repeating each test 100 times. In addition, it was verified that in all experiments the same results were obtained for the same comparisons.

Figure 1 (a) summarizes the results obtained. It should be noted that for the experiments the number of simultaneous queries was set to 56 in order not to exceed the occupancy factor of the GPUs used. As can be seen in both cases the processing times decrease as the number of nodes in the system increases, with the multi-GPU algorithm being 1.6 times faster than the multi-node/multi-core algorithm.



(a) Processing time (seconds). (b) Speed-up

Fig. 1: Processing time and speed-up when solving kNN queries (k=32) for the multi-GPU algorithm versus the multi-node/multi-core algorithm for 56 simultaneous queries and 4,000,000 items. The X-axis shows the number of nodes and also the number of GPUs.

Figure 1b shows the speed-up for the multi-GPU and multi-node/multi-core algorithms. It should be clarified that for the calculation of the speed-up (Time(Sequential)/Time(Parallel)), the execution time of the sequential version on CPU was taken as a reference. From the results obtained it is observed that the Speed-up increases as the number of nodes increases for both cases, for 4 processing nodes the multi-GPU algorithm has a Speed-up of 159.428x and 87.057x for the multi-node/multi-core algorithm. The detailed values are expressed in Table 1 along with processing time values in seconds of the algorithms.

5 Conclusions

This paper proposes an implementation on two types of multi-GPU and multi-node/multi-core platforms, for massive searches on unstructured databases. The presented method aims to reduce the time involved in the search process when faced with a large number of simultaneous queries on the system.

Table 1: Processing time (Tp) in seconds and speed-up (Sp) of multi-GPU and multi-node/multi-core algorithms.

Nodes/GPU	muti-GPU		multi-node/multi-core	
	Tp	Sp	Tp	Sp
1	77,097	43,893	134,974	25,071
2	39,344	86,011	68,045	49,732
3	39,687	85,267	49,530	68,322
4	21,226	159,428	38,871	87,057

Experimental validation shows that the multi-GPU approach is 1.6 times superior to the multi-node/multi-core algorithm in solving 56 simultaneous queries. Moreover, in both algorithms the speedup is directly proportional to the number of nodes (number of GPUs) reaching 156x for 4 GPUs and 87x in the case of the hybrid multi-node/multi-core algorithm.

In future work, we plan to increase the number of elements in the database, with the goal of reaching 16 million individuals. Using a database of this size brings with it the problem of the overall memory capacity of the GPU, being necessary to explore a multi-node/multi-GPU approach. Also, we will explore others distribution strategies for the multi-core platform algorithm.

References

1. Barrientos, R.J., Gómez, J.I., Tenllado, C., Prieto Matias, M., Marin, M.: Range query processing on single and multi GPU environments. Computers & Electrical Engineering 39(8), 2656–2668 (Nov 2013), http://www.sciencedirect.com/science/article/pii/S0045790613001560
2. Barrientos, R., Gómez, J., Tenllado, C., Prieto, M., Marin, M.: knn query processing in metric spaces using gpus. In: 17th International European Conference on Parallel and Distributed Computing (Euro-Par 2011). Lecture Notes in Computer Science, vol. 6852, pp. 380–392. Springer, Bordeaux, France (September 2011)
3. De Battista, A., Pascal, A., Gutiérrez Retamal, G.A.: Un nuevo índice métrico-temporal: el historical FHQT (2007), http://hdl.handle.net/10915/22138
4. Gil-Costa, V., Marin, M., Reyes, N.: Parallel query processing on distributed clustering indexes. Journal of Discrete Algorithms 7(1), 3–17 (Mar 2009), http://www.sciencedirect.com/science/article/pii/S1570866708000749
5. Guidet, S., Barrientos, R.J., Hernández-García, R., Frati, F.E.: Exhaustive similarity search on a many-core architecture for finger-vein massive identification. Journal of Physics: Conference Series 1702(1), 012012 (Nov 2020), https://doi.org/10.1088/1742-6596/1702/1/012012, publisher: IOP Publishing
6. Hernández-García, R., Guidet, S., Barrientos, R.J., Frati, F.E.: Massive finger-vein identification based on local line binary pattern under parallel and distributed systems. In: 2019 38th International Conference of the Chilean Computer Science Society (SCCC). pp. 1–7. IEEE (2019)
7. Nations, U.: Macrodatos para el desarrollo sostenible | Naciones Unidas, https://www.un.org/es/global-issues/big-data-for-sustainable-development, publisher: United Nations
8. Rosdi, B.A., W.Shing, C., Suandi, S.A.: Finger vein recognition using local line binary pattern. Sensors 11, 11357–11371 (2011)