Performance comparison of an implementation of a variant of the binary search algorithm with gap-encoding and search executions parallelization

Alan Keith Paz
Magíster en Informática
Universidad Austral de Chile
Valdivia, Chile
alan.keithpaz@gmail.com

***Abstract*—In this paper it's compared the performance of some implementations of binary search algorithms, where HPC techniques have been applied to improve both the storage space of the data structures (for faster access) and their execution, as well as part of the general code has been parallelized. In linear search the time complexity of the algorithm is O(n), a binary search reduces the time complexity to O(Log n). In general terms, a binary search takes a sorted array and searches in it by repeatedly dividing the search interval in half. For these comparisons we use 4 implementations, bsearch() C programming language Binary Search (AutobSearch) and 3 iterative methods: Binary Search (BS), Binary Search with a Sample Scan (BSSCN) and a Gap-Encoding Binary Search (BSGSCN).**

The result shows that an improvement in these dimensions can significantly influence the final execution time, so it is a good place to start, thinking about the impact on possible applications.

Keywords—binary search, gap-encoding, parallel computing, High Performance Computin.

I. Introduction

The challenge of looking for ways to improve the timing and the performance of a binary search in an ordered vector (array of a dimension) arises from the motivation in the search to apply High Performance Computing (HPC) techniques not only paralleling, but also analyzing the algorithms and searching they the opportunity.

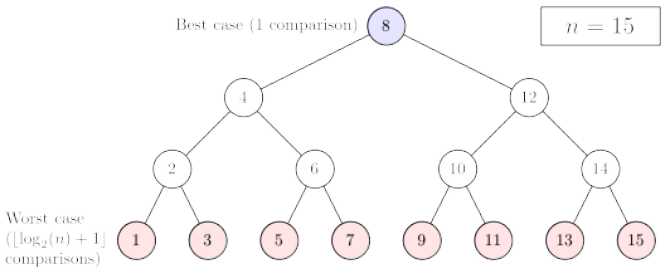

Fig. 1: A illustration of the complexity of the binary search algorithm.

The binary search is a widely used and basic algorithm when it comes to some more complex implementations. Although the binary search requires that the array be ordered, it is much faster than a linear search and we must remember that it applies to a wider range of problems than hash tables. Nowadays we find several implementations of binary search (binary search trees, exponential search, interpolation search, fractional cascading, etc.) and with multiple applications (computational geometry, machine learning, etc.)

To carry out this study, we propose a storage implementation of gap-encoding and parallel execution of binary searches. We compared array creation times, the efficiency of the algorithms implemented as a function of the execution time and the accuracy (in comparison with the different implementations of algorithms).

To make execution faster, we look for parallelizable segments, identifying the execution blocks associated with the repetitions, as well as the construction of each matrix associated with each storage method. The result shows that an improvement in these dimensions can significantly influence the final execution time. so it's a good place to start, thinking about the impact on the possible applications.

II. Related Work

We can consider the publications found as an informative bibliography, however there is not much about the parallel implementation of the binary search with one-dimensional arrays, since it is unattractive to say the least (Parallel implementation can speed up a binary search, but the improvement is not particularly significant, in worst case, the time required for a binary search is log2 (n), a simple parallel implementation breaks the master list into k sub-lists to be bin-searched by parallel threads. Time for the binary search is log2 (n/k) realizing a theoretical decrease in the search time, but this increase in speed is affected by a very large increase in the number of operations to be performed, so the use of a improvement in efficiency).

Instituto de Informática, Facultad de Ciencias de la Ingeniería, Universidad Austral de Chile

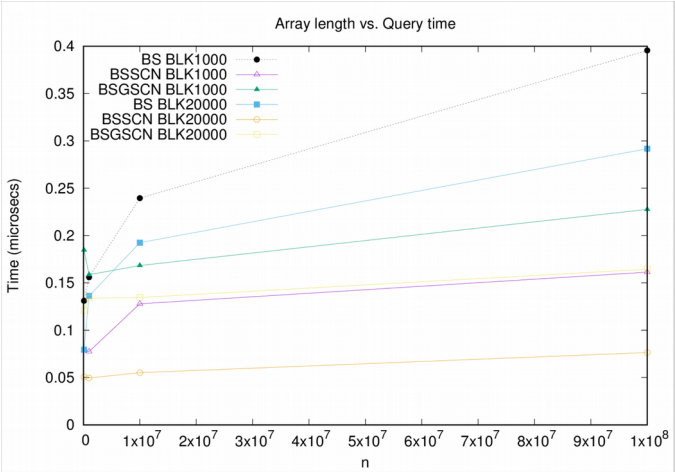

Fig. 2: Execution time with array dimension increments. Binary Search (BS), Binary Seach Sample Scan (BSSCN), GAP-ENCODING BINARY SEARCH (BSGSCN).

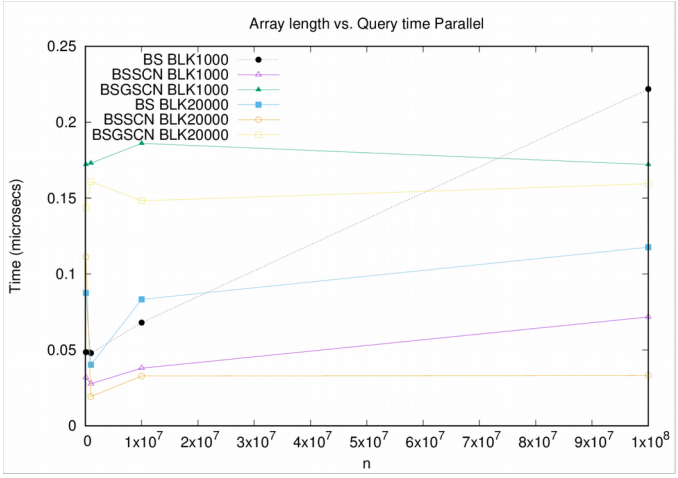

Fig. 3: Execution time with array dimension increments for Parallel Improvements Version. Binary Search (BS), Binary Seach Sample Scan (BSSCN), GAP-ENCODING BINARY SEARCH (BSGSCN).

About the parallel implementation of binary search, we can delve into a case of search of values between two arrays in the publication of Akl S. and Meijer H. [1]. However, we can find the use of these searches in multiple applications, such as the case of seeing the development of applying parallel computing to problems of serial algorithms in work, carried out by Megiddo N. [2] as early as the 1980s; Chen D. 1995 delves into the efficient use of binary searches and their applications [3]; use of improved binary searches for anti-collision algorithms [4]; its use for convolutional neural network training [5]; Hybrid search algorithms [6], among others.

III. Implementation and Measures

The implementation of the algorithm has been done in the programming language C, which also has a binary search calculation method that is evaluated as well as the own implementations. Regarding the particularities, with regard to the storage of the data structures, the data type is used reduced to the upper limit size of the data, this means that for all the elements in an X[] array we take the maximum value in X and we use the number of bits necessary to represent it to establish the space for each element of it. This method was introduced during the course of HPC in the Master of Informatics of UACh by the Phd. Héctor Ferrada.

For the creation of the data for tests a normal distribution and a uniform distribution are used, establishing the preference at the time of running the program.

The times as the size of the array grows and how this influences the search for non-paralyzed iterative versions of the algorithms can be seen in Fig. 2. We can see that as the chaff of the array grows, the search based on sapling and scan (BSSCN) remains constant, however the binary search method without improvements (BS) experiences a significant increase in time as the array grows.

In Fig. 4 we can see the times obtained for the different tests carried out for structure creation.

By implementing the possible parallelism improvements in the program (parallelization in the creation of structures, parallelization in execution of multiple searches), better times and a notable increase in performance are obtained. In Fig. 3 we can see array length vs query time for parallel version. The corresponding Speedup and Efficiency measures are found in Fig. 5, Fig. 6 and Fig. 7

The variant with gap-encoding storage only has better performance when working with larger arrays, when it is used with smaller arrays its performance is less than that of the sequential algorithm without improvements, taking longer to perform searches, but always with a lot of smaller storage space. These can be seen in Fig. 8.

TABLE I. Times Elapsed For N=10^8

# thread	Times for n = 10^8, in Microseconds		
	Structures Creation	***AutoBS bsearch()***	***Gap-encoding BS***
1	1,246893	0,387567	0,205316
2	0,790619	0,297915	0,172597
3	0,771005	0,155244	0,0920431
4	0,707087	0,155772	0,119381
5	0,794974	0,166735	0,0879803
6	0,757375	0,141201	0,0841744
7	0,752468	0,133641	0,131282
8	0,728626	0,116514	0,128219

a. Measeares taken during executions

Fig. 4: Times observed using a CPU Intel Core i5 at 2.50GHz with 2 physical cores, 4 logical cores, 6 GB RAM DDR3 at 1333 MHz on Debian 9.

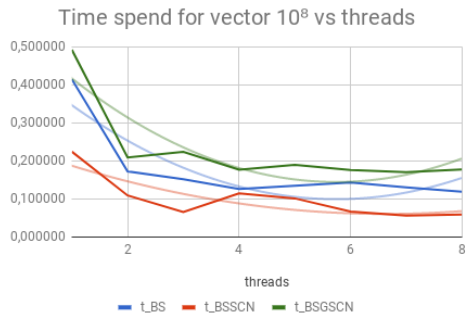

Fig. 5: Time spend for each Binary Search Alternative

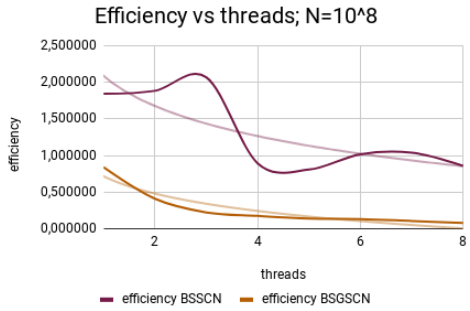

Fig. 6: Efficiency measure for BSSCN and BSGSCN

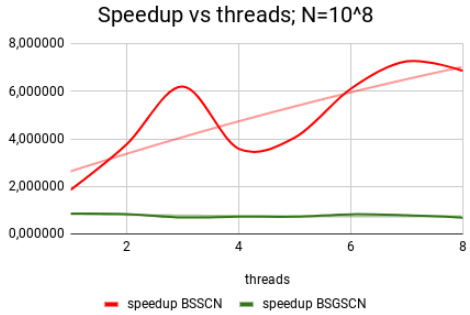

Fig. 7: Speedup measure for BSSCN and BSGSCN

TABLE II. MEMORY USED FOR EACH ARRAY

N	Memory Size			
	Original Array (A[])	***Reduced Array X[]***	***GAP Array G[]***	***Sample Array for G, S[]***
10^4	0,07629 MiB	0,02027 MiB	0,01550 MiB	0,00184 MiB
10^5	0,76293 MiB	0,20266 MiB	0,13113 MiB	0,01192 MiB
10^6	7,62939 MiB	2,02656 MiB	1,19209 MiB	0,08444 MiB
10^7	76,2939 MiB	20,2656 MiB	15,4972 MiB	0,61411 MiB
10^8	762,939 MiB	202,656 MiB	143,051 MiB	4,60582 MiB

Fig. 8: Memory Size observed using each storage method.

IV. DISCUSSION AND CONCLUSION

The implementation shows that we can improve theoretical and practically the efficiency of this algorithm using HPC techniques for best manage of memory storage, as using parallel coding for accelerate the execution and the stages of the program.

The most difficult part of this job it is about choose the right segments for parallel coding, specially when it's about iterative algorithms, because the "factibility barrier" it is so important as velocity, for keep integrity on the code and accuracy on the results.

A parallel implementation of Binary Search based on dividing the array in the number of threads or processors to ensure the optimal utilization and optimization (thinking that can improve even a 30% the velocity) it would be pending for a next level of this work.

REFERENCES

[1] *S. G. Akl and H. Meijer, "Parallel binary search," in IEEE Transactions on Parallel and Distributed Systems, vol. 1, no. 2, pp. 247-250, Apr 1990. doi: 10.1109/71.80139.*

[2] Nimrod Megiddo. 1983. Applying Parallel Computation Algorithms in the Design of Serial Algorithms. J. ACM 30, 4 (October 1983), 852-865. DOI=http://dx.doi.org/10.1145/2157.322410

[3] D. Z. Chen, "Efficient parallel binary search on sorted arrays, with applications," in IEEE Transactions on Parallel and Distributed Systems, vol. 6, no. 4, pp. 440-445, Apr 1995. doi: 10.1109/71.372799

[4] Y. Bai, L. Yang, G. Zhang and Y. Xu, "An improved binary search RFID anti-collision algorithm," 2017 12th International Conference on Computer Science and Education (ICCSE), Houston, TX, 2017, pp. 435-439. doi: 10.1109/ICCSE.2017.8085531

[5] D. Lunga, L. Yang, J. Yuan and B. Bhaduri, "Hashed binary search sampling for convolutional network training with large overhead image patches," 2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), Fort Worth, TX, 2017, pp. 767-770. doi: 10.1109/IGARSS.2017.8127065

[6] A. E. Jacob, N. Ashodariya and A. Dhongade, "Hybrid search algorithm: Combined linear and binary search algorithm," 2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS), Chennai, 2017, pp. 1543-1547. doi: 10.1109/ICECDS.2017.8389704