A Parallel Search Algorithm for the SAT

V. Gil Costa, A. M. Printista, N. Reyes
University of San Luis
San Luis, Argentina
{gvcosta,mprinti,nreyes}@unsl.edu.ar

M. Marín
Computing Department
University of Magallanes
Punta Arenas, Chile
mmarin@ona.fi.umag.cl

Abstract. In order to be able to perform multimedia searches (like sounds, videos, images, etc.) we have to use data strucures like the Spatial Approximation Tree (SAT). This structure is a nice example of a tree structure in which well-known tricks for tree parallelization simply do not work. It is too sparse, unbalanced and its performance is too dependent on the work-load generated by the queries being solved by means of searching the tree. The complexity measure is given by the number of distances computed to retrieve those objects close enough to the query. In this paper we examine some alternatives to parallelize this structure through the MPI library and the BSPpub library.

Key-Words: SAT, metric spaces, searches, MPI, BSP

1 Introduction

Data parallelism is one of the most successful efforts to introduce explicit parallelism to high level programming languages. The approach is taken because many useful computation can be framed in terms of a set of independent sub-computations, each strongly associated with an element of a large data structure. Such computations are inherently parallelizable. Data parallel programming is particulary convenient for two reasons. The first, is its easiness of programming. The second is that it can scale easily to large problem sizes.

One of these problems is the search in metric spaces by spatial approximation. A metric space is formed by a collection of objects and distance function defined among them, which satisfies the triangle inequality. The goal is given a set of objects and a query, retrieve those objects close enough to the query. The Spatial Approximation Tree (SAT) is a data structure devised to support efficient searching in high-dimensional metric spaces [5, 6]. It has been compared successfully against other data structures [2, 3] and update operations have been included in the original design [1, 7].

Some aplications for the SAT are non-traditional databases (e.g. storing images, fingerprints or audio clips, where the concept of exact search is not used and we search instead for similar objects); text searching (to find words and phrases in a text database allowing a small number of typographical or spelling errors); information retrieval (to look for documents that are similar to a given query or document); etc.

A typical query for this data structure is the $range\ query$ which consists on retrieving all objects within a certain distance from a given query object. The distance in a high-dimensional space is expensive to compute and is usually the relevant performance metric to optimize, even over the secondary memory operation cost [1]. This problem is more significant in very large databases, making it relevant to study efficient ways of

parallelization.

In this paper we propose three parallel algorithms for range query operations for the SAT data structure using the MPI library [8] and the BSPpub [11].

2 Sequential SAT

The SAT construction starts by selecting at random an element a from the database S. This element is set to be the root of the tree. Then a suitable set $N(a)$ of neighbors of a is defined to be the children of a. The elements of $N(a)$ are the ones that are closer to a than any other neighbor. The construction of $N(a)$ begins with the initial node a and its bag holding all the rest of S. We first sort the bag by distance to a. Then we start adding nodes to $N(a)$ (which is initially empty). Each time we consider a new node b, we check whether it is closer to some element of $N(a)$ than to a itself. If that is not the case, we add b to $N(a)$. We now must decide in which neighbor's bag we put the rest of the nodes. We put each node not in $a \cup N(a)$, but in the bag of its closest element of $N(a)$. The process continues recursively with all elements in $N(a)$.

The resulting structure is a tree that can be searched for any $q \in S$ by spatial approximation for nearest neighbor queries. Some comparations are saved at search time by storing at each node a its covering radius, i.e., the maximum distance $R(a)$ between a and any element in the subtree rooted by a.

Range queries q with radius r are processed as follows. We first determine the closest neighbor c of q among $\{a\} \cup N(a)$. We then enter into all neighbors $b \in N(a)$ such that $d(q, b) \leq d(q, c) + 2r$. This is because the virtual element q sought can differ from q by at most r at any distance evaluation, so it could have been inserted inside any of those b nodes. In the process we report all the nodes q we found close enough to q. Finally, the covering radius $R(a)$ is used to further prune the search, by not entering into subtrees such that $d(q, a) > R(a) + r$, since they cannot contain useful elements.

```
1:  procedure RANGESEARCH(Node a, Query a, Radius r)
2:      if (d(a,q) ≤ R(a)+r) then
3:          if (d(a,q)≤ r) then
4:              report a
5:          end if
6:          d_min ← min{d(c,q),c ∈ {a}∪ N(a)}
7:          for (b ∈ N(a)) do
8:              if (d(a,q)≤ d_min+2r) then RangeSearch(b,q,r)
9:              end if
10:         end for
11:     end if
12: end procedure
```

3 Experimental Setup

The implementations of the proposed strategies were performed using the MPI and the BSPpub libraries. The database used in our experiments is a 69K-word English dictionary and queries are composed by words selected uniformly at random. The distance is the edit distance, that is, the minimum number of character insertions, deletions, and replacements to make two strings equal. We assume a demanding case in which each query has one word and the search is performed with four different radius (1,2,3 and 4).

For the parallelization of the SAT, we assume a server operating upon a set of P machines, each containing its own memory. Clients request service to a $broker$ machine,

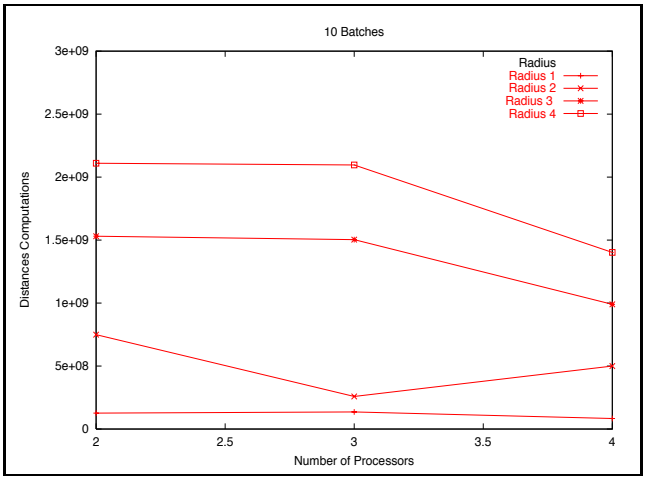

Fig. 1: The copy approach for 10 batches of queries.

which in turn distributes those requests evenly onto the P machines implementing the server. Requests are queries that must be solved with the data stored on the P machines. We assume that under a situation of heavy traffic the server start the processing of a batch of Q queries.

4 Parallel SAT

Now we will be describe and we will analyze three strategies that can be used for the parallelization of the SAT data structure.

A first, but native, approach to parallelization is simply assume that the processors has enough memory to maintain each one a complete copy of the SAT data structure. In this case the queries are distributed evenly onto the processors and their processing is straightforward as we just apply the sequential algorithm locally. No inter-processors communication is required and every query can be solved in just one step.

Fig. 1 shows results for this strategy with a search radius 1,2,3 and 4, $P = 4$ processors and 10 batches of queries. Here the SAT is initialized with the 90% of the dictionary words and the remaining 10% are left as query objects (randomly selected from the whole dictionary). As the radius grows up also does the number of distances computations, because we can compare the queries elements with more objects of the SAT than with a small radius.

Then the Fig. 2 and the Fig. 3 shows the results for a search with radius 1 and 2, the experiments performed with radius 3 and 4 have the same behavior presented in these figures. Here as the percentage of the database is increased, the queries search cost is reduced because the number of queries is smaller. But this strategy presents fluctuations where the maximum number of distances computations are performed with the 50% of the database. These graphs show how dependent is this strategy on the query distribution. Apart from the over-consumption memory problem, this strategy is not convenient since it in fact is not able to achieve a good performance.

Therefore, this strategy does not present a parallelism overhead, but it neither can scale, because to build the SAT tree it is necessary that each processor has the complete database in its local memory, and as the database grows up, the construction cost also does.

Finally, the workload of the processors will depend on the queries that each one

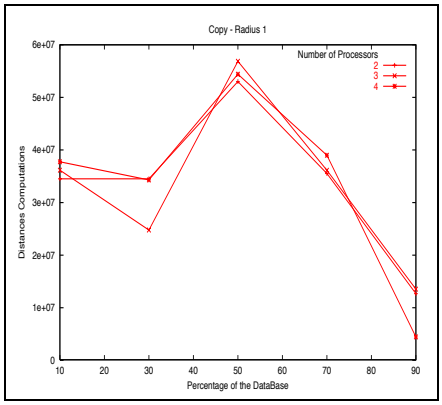

Fig. 2: Radius 1.

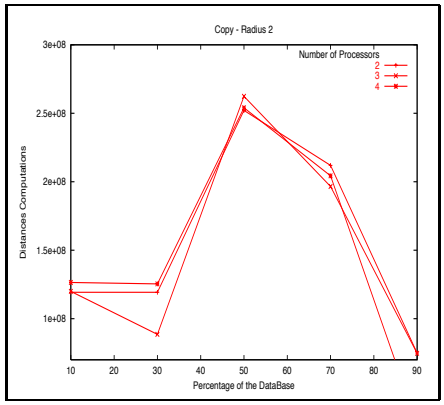

Fig. 3: Radius 2.

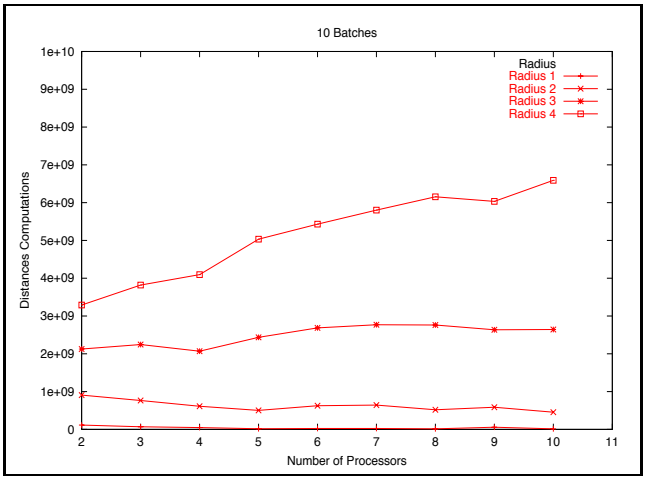

Fig. 4: The random approach with the 90% of the database.

gets, since all they perform the same operations. In the following we propose a different approach to parallelize the SAT data structure.

A second approach to parallelize the SAT, would be to divide the problem in parts and to distribute them among the processors of the server. In this second strategy, a type of well-known partition is used named domain decomposition is used, where the data associated to the problem are divided and then each parallel task works on a portion of these data. Here the database is divided and distributed among the processors at random. Once each processor gets its portion of the database, it can begin to build the SAT structure using its local data.

All the queries go to the processors, because the objects can be from different data types (sound, text, videos, etc.), and we do not know which processor has information for the query.

Next, the Fig. 4 shows the results obtained for the execution of 10 batches with this strategy using up to 10 processors, with a radius between one and four. With a small radius, as the number of processors is increased the cost is reduced, but with a bigger radius this behavior changes, due the processors has to explore a bigger space to find the answers.

In this strategy, as in the previous one, it is not necessary to send and to receive messages during the search process over the tree, because each processor works using

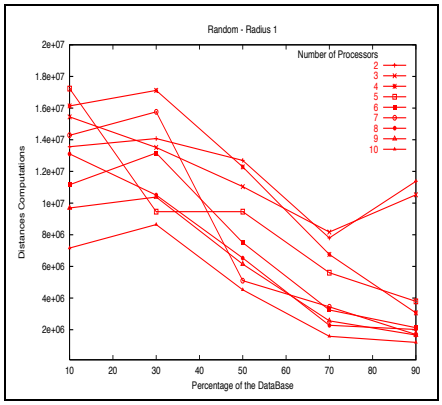

Fig. 5: Radius 1.

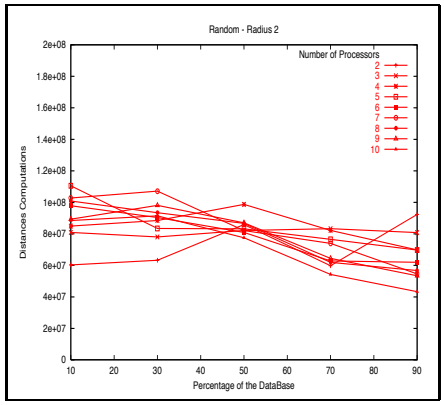

Fig. 6: Radius 2.

the tree stored in its own memory locally. Communication only exists at the beginning of the search operation, when the queries are sent to the processors. But contrary to the previous case, the communication is bigger, because the queries are sent to the machines (broadcast). The goal of using a random distribution of the database, is to minimize the unbalance presented during the queries search, since the number of distances computations in each processor will depend on the data that it has received to build its SAT tree, and on the query that is being processed.

The Fig. 5 and Fig. 6 show the results for this second strategy (Random). With a small radius we can see that as the number of processors is bigger we have less distances evaluations. Now with a bigger radius if we use more than the 50% of the database this holds; but with less than the 50% of the database the results are very dependent on the queries.

Finally we present the last strategy for the SAT data structure, where not only the database an the queries are distributed through a hash function. This kind of strategies may be use to perform searches in a Web dictionary, where the queries search are not exact. That is to say, when the system has to find similar words. So this strategy can be used when the database from where the objects of the metric space are obtained, is formed by text (words).

The Fig. 7 shows the results for this strategy with the 90% of the database, and the Fig. 8 and Fig. 9 show the results as the percentage of the database is increased.

In this case we have a more concentrated space, because all the similar words from the dictionary go to the same processor, and that implies a harder search metric space due the triangle inequality permits discarding less elements.

5 Comparation of the Strategies

The Fig. 10 and Fig. 11 show the distance evaluations obtained by each strategy with $P = 4$ processors as the database size is increased. In all the cases (working with radius 1,2,3 and 4), the second strategy shows a better performance than the others. The reason why the random strategy presented allows to reduce the number of distance evaluations is because the elements of the database are spread over the processors, and each one will have an easy space where the elements are more sparse, and in this case the SAT structure performs better than in a hard space [5].

Lastly the Fig. 12 shows the results for the copy strategy with four processors, the random with ten processors and the hash with eight processors. Here you can see that

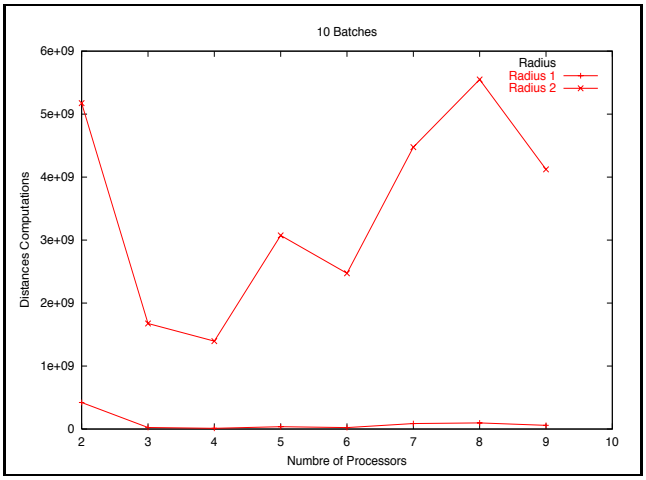

Fig. 7: The hash with the 90% of the database.

the random strategy outperform the other two, while the hash strategy works well with a big database size, where each processor can work with more different elements.

6 MPI vs. BSPpub

In this paper we examine some alternatives to parallelize this structure first with the MPI library and then using the BSPpub library. The BSPpub library is oriented to the BSP model [11] and offers, the same basic functionality as BSP. The essence of the BSPpub focus to the parallel programming is the $supersteps$ concept.

The computation is organised as a sequence of supersteps. During a superstep, the processors may perform sequential computations on local data and/or send message to others processors. The message are available for processing at their destination by the next superstep, and each superstep is ended with the barrier synchronization of processors.

In the BSP model any parallel computer is seen as composed of a set of P processor-local-memory components which communicate with each other through messages. A BSP computer is characterized by the bandwidth of the network, the number of processors, its speed and for the synchronization time among all the processors. All these characteristics are part of the parameters of a BSP computer.

The BSP model establishes a new style of parallel programming to write programs of general purpose, whose main characteristic are its easiness and writing simplicity, its independence of the underlying architecture (portability). BSP achieves the previous properties elevating the level of abstraction with which the programs are written.

The differences presented among both implementations are exclusive of the characteristics that each library has, and how the programming model adapts to the problem.

With MPI, we do not need to synchronize the processors, so when any machine finish a batch it can continue with the next batch without wasting time. While BSPpub works with supersteps and at the end of each one there is a barrier synchronization, making that every processor has to wait for the others to continue with the next batch of queries. In a real system where the time is important it will be harmful.

In our research for parallelizing the SAT structure we use the number of distances evaluations as the complexity measure, so the results obtained for each strategy are independent of the library (MPI or BSPpub) that we used.

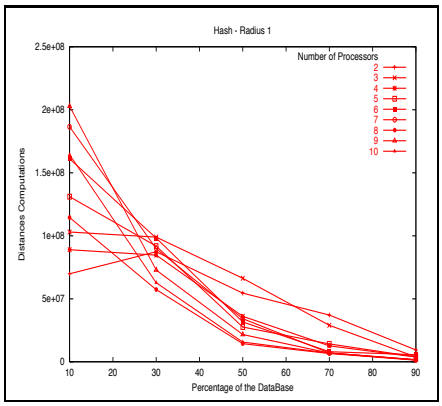

Fig. 8: Radius 1.

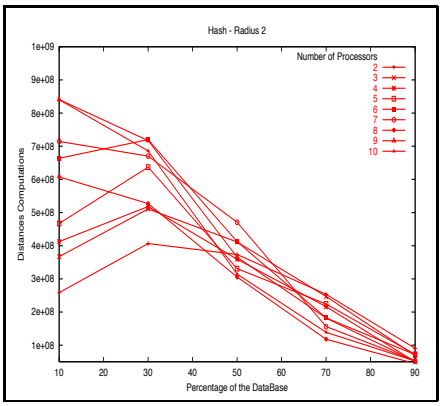

Fig. 9: Radius 2.

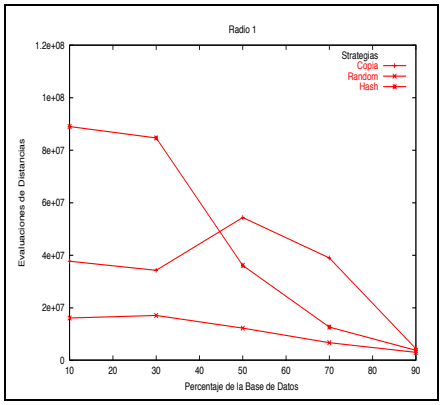

Fig. 10: Radius 1.

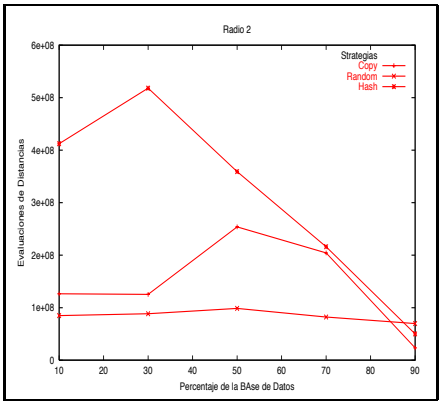

Fig. 11: Radius 2.

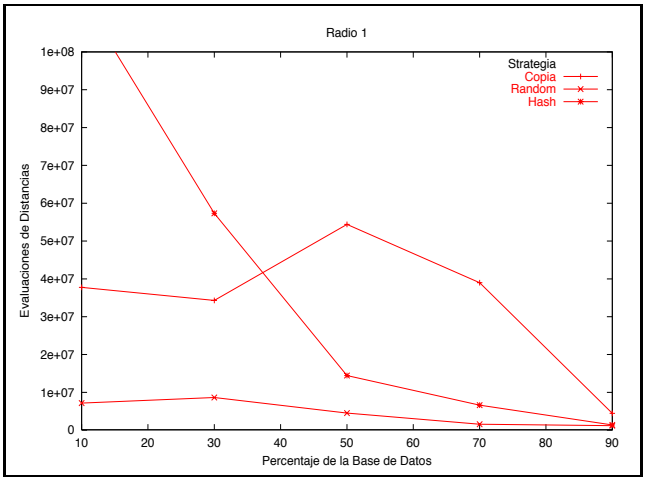

Fig. 12: Best results obteined.

An advantage presented by BSPpub, is that it has a computing model associated that will allow to predict the costs of the algorithms theoretically.

So to select the best library for our problem we should evaluate the functional requeriments and the execution environment of the application to choose the API that has the wanted characteristics.

7 Final Comments

All presented strategies don't have data dependence, since each processor has a copy of the SAT, or it builds its own tree using a portion of the database. Consequently none of these strategies requires a synchronization barrier at the moment to exchange data (when the queries are sent).

Of the results obtained in each presented strategy, we can observe that as the search radius is increased also does the number of distances computations.

With regard to the libraries, it is convenient to choose the MPI if this problem will be use in real time systems, but if it will be used for research reasons it's convenient to use the BSP model to obtain theoretical predictions.

A point to emphasize is that the SAT structure has nodes with a diverse number of children, and each son can cause a distance comparison. Therefore it is important to be able to balance the number of comparisons of distances performed in each processor. Then, it is desirable map the nodes of the tree among the processors considering the distance comparisons that they can be potentially performed in each subtree.

As future work, we are trying to find a strategy that allows a better balance work and reduce the compute performed in each processor by increasing the communication. We are talking about a dynamic mapping to distribute the workload.

References

[1] D. Arroyuelo, F. Muñoz, G. Navarro, and N. Reyes. Memory-adaptative dynamic spatial approximation trees. In Proceedings of the 10th International Symposium on String Processing and Information Retrieval (SPIRE 2003), LNCS 2857, pages 360-368. Springer, 2003.

[2] C. Bohm, S.Berchtold, and D. Kein. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. ACM Computing Surveys, 33(3):322-373,2001.

[3] V. Gaede and O. Gunther. Multidimensional access methods. ACM Computing Surveys, 30(2):170-321,1998.

[4] M. Marín and G. Navarro. Distributed query processing using suffix arrays. In Proceedings of the 10th international Symposium on String Processing and Information Retrieval (SPIRE 2003), LNCS 2857, pages 311-325. Springer, 2003.

[5] G. Navarro. Searching in metric spaces by spatial approximation. The Very Large Databases Journal (VLDBNJ), 11(1):28-46, 2002.

[6] G. Navarro and N. Reyes. Fully dynamic spatial approximation trees. In Proceedings of the 9th International Symposium on String Processing and Information Retrieval (SPIRE 2002), LNCS 2476, pages 254-270. Springer, 2002.

[7] G. Navarro and N. Reyes. Improved deletions in dynamic spatial approximation trees. In Proc. of the XXIII International Conference of the Chilean Computer Science Society (SCCC'09), pages 13-22, IEEE CS Press, 2003.

[8] Snir M., Otto S., Huss-Lederman S., Walker D., Dongarra J. MPI: The complete Reference, Cambridge MA: MIT Press, 1996.

[9] L. Valiant. A Bridging Model for Parallel Computation. Communications of the ACM, Vol. 33, Pp 103-111, 1990.

[10] WWW.BSP and Worldwilde Standard, http://www.bsp-worldwide.org

[11] WWW.BSP PUB Library at Paderborn Univertity,