Research Journal of Applied Sciences 6 (1): 70-75, 2011
ISSN: 1815-932X

Parallel Sequential Searching Algorithm for Unsorted Array

Ahmad Haboush and Sami Qawasmeh
Department of Computer Science, Faculty of Science, Jerash University,
P.O. Box 311, 26110 Jerash, Jordan

Abstract: Parallel search is a way to increase search speed by using additional processors. Researchers propose a parallel search algorithm that searches an item in unordered array, the searching time obtained is better than that obtained in binary search. That is justified by the fact that the binary search requires a variant time for sorting the input array. The speed up of the proposed algorithm is increased linearly with the input size by saving the time spent in sorting input data. In the proposed algorithm, the array to be searched is divided into two subarrays and then, two search threads are created in parallel which required O (n/2) in the worst case (where all the items is scanned), reducing the searching time in the worst case to O (n/2+log n). Log n is the time needed to splitting an array of size n. The efficiency is increased quickly for an input size of 5000-1,000,000 item. However, the efficiency suffers a little variation for an input size of 1,000,000-5,000,000 item that is because the binary search has an optimal running time for large sorted input size where the searching space is reduced by a factor of 2 each time.

Key words: Parallel computing, searching algorithms, binary search, parallel multithread search, sorting efficiency, splitter-based searching, efficiency

INTRODUCTION

In the information age, the huge amount of information and data available and exchanged, require a fast and efficient searching mechanisms for effective use of this information. Generally, to find an item in an unordered array of size n, it's used one of the following approaches: Scans all the values that require O (n) time. Using a sorting algorithm like (Quicksort, Bubble sort, Binary Tree sort...) (Williams, 1964; Knuth, 1997; Cormen *et al*., 2001) that varies in complexity from O (n log n) in the best case to O (n^2) in the worst case. Moreover, the memory usage for searching these items varies from O (1) to O (log n).

Using Heapsor that performs better than Quicksort or other sorting algorithms, it takes O (n log n) in the worst and best case (Williams, 1964; Knuth, 1997; Cormen *et al*., 2001). In cases (2, 3), a searching mechanism is used subsequently to find the location of the searched item. Searching algorithms are techniques used to make the searching of an information in any field fast and more efficient. Searching problem is defined as follow: given an input x determines whether there exists a y such that f (x, y) is true.

Binary search is an algorithm for locating the position of an element x in a sorted list. It starts by dividing the array into subarrays L, R. Then, a comparison between the value of x and the values of the first elements in each subarray is done and to define in which subarray the searching process must start. This procedure (splitting and comparing) continues until finding the requested item (this procedure continue at most log n time). The idea on which the binary search algorithm based on is to reduce the searching space each time by a factor of two, the worst case performance of binary searching is log n for an input array of size n (Knuth, 1997; Cormen *et al*., 2001; Williams, 1964).

Parallel search or multithread search (SMP) is a way to increase search speed by using additional processors. Utilizing these additional processors is an interesting domain of research. SMP algorithms are classified by their scalability (that means the behavior of the algorithm as the number of processors become large) and their speed up. The speed up is defined as the ratio of the running time of the sequential execution to the running time of the parallel execution. It is mostly used as an indicator for a parallel program's performance (Cormen *et al*., 2001).

In this study, we propose a new parallel search algorithm that performs better than the binary search in terms of running time. The algorithm works on an unsorted array saving in that the time spent in sorting. In binary search algorithm, the time needed to search an item is the time needed for sorting+time needed for splitting+searching time. That is equal to (n+2) log n in

Corresponding Author: Ahmad Haboush, Department of Computer Science, Faculty of Science, Jerash University, P.O. Box 311, Jerash 26110, Jordan

the best case and O (n^2+2) log n in the worst case. In the algorithm, initially the array is divided into two subarrays and then two processors search them in parallel that require O (n/2) in the worst case (where all the items is scanned), reducing the searching time in the worst case to O (n/2+ log n). Log n is the time needed to splitting an array of size n. To insert an item in sorted array, the correct location must be found. In the algorithm the item can be inserted in any location. It outperforms the performance of binary search in the case where n is an odd number. In this case, the array is split into two subarrays of size (n-1/2), n/2. However, in binary search the problem of odd number continues for each splitting operation until finding the searched item.

Multithreaded search SMP has been gaining popularity recently with the availability of multiprocessor computers. It is widely used in traversing a search tree. In (Koorangi and Zamanifar, 2007), it is applied the parallelism on genetic algorithm to find an optimization methods. The genetic algorithm is appropriate for this purpose because it is independent of the primary values of a system and because it is independent of the system's objective function properties. In addition, it allows searching of greater space of the parameters values.

Binary search tree is a binary tree whose nodes are organized according to the binary tree property. The time required to search a given key can vary from tree to tree depending on the depth of the node where the key is found or on the length of the branch searched. An optimal binary search tree is a binary search tree with minimum expected comparisons. Depend on special set of keys (C_i) and their possibilities (P_i), the optimization of binary search tree in minimizing the relation $\Sigma^n_{i=1}\ C_iP_i$. The parallel genetic algorithm is used to reduce the time needed to find an optimal solution for binary search tree because considered the function $C_i\ P_i$ as fitness function and the chromosomes of the genetic population is used as keys.

In executing of the genetic algorithm each chromosome shows a value of k its length should not exceed a determined length depending on the number of keys. Also, the maximum of the genetic population for each processor must be defined. The type of cross over used is the common single point method. For creating the intermediate population and for selection of the parents it is used the Roulette Wheel method. The algorithm execution has time cost O (n) ignoring the genetic algorithm time. If a processor is allocated a process, the cost will be O (n^2/N). Experimental results show that the speed-up is increased with the number of processors. Chaslot *et al.* (2008) discussed the use of three parallelism methods are leaf parallelism, root parallelism (Cazenave and Jouandeau, 2007) and tree parallelization method for Monte Carlo Tree Search (MCTS). MCTS (Coulom, 2007; Kocsis and Szepesvari, 2006) is a best first search method that does not require a positional evaluation function. It is based on randomized exploration of the search space. In leaf parallelization (Cazenave and Jouandeau, 2007) is used on thread that traverses the tree and adds one or more nodes to the tree until reaching leaf. Then, starting from the leaf simulated games are played for each available thread. When all games are finished, the results of all these simulated games are propagated backwards through the tree by one single thread.

In this technique, the time required for simulated game is unpredictable to decrease the waiting time, the program might stop the simulations that are still running when the results of the finished simulation become available. In root parallelization (Cazenave and Jouandeau, 2007), multiple MCTS tree are built in parallel with one thread for tree. The threads do not share information with each other. When the available time is spent all the root children of the separate MCTS tree are merged with their corresponding clone. For each group of clones, the scores of all games played are added.

The best move is selected based on this grand total. This parallelization method only requires a minimum amount of communication between threads. So, the parallelization is easy even on a cluster. In tree parallelization (Chaslot *et al.*, 2008) is used one shared tree from which several simultaneous games are played. Each thread can modify the information contained in the tree. Mutexes (mutual exclusion) are used to lock from time to time certain parts of the tree to prevent data corruption. To evaluate the performance of the tree parallelization it is used the Games-Per-Second speed UP (GPS) that is computed by dividing the number of simulated games per second performed by multithreaded program by the number of games per second played by a single thread program. The results show that for leaf parallelization, the GPS speedup is increased with the number of threads is quite low. However, the root parallelization is a quite effective way of parallelizing MCTS.

But for four processors threads, the strength speed up is significantly higher than the number of threads used. For tree parallelization the GPS speed up obtained is satisfied. However, the strength speed up obtained up to four threads is satisfactory but for 16 threads it is insufficient.

Gayatri and Baruah (2008) proposed a parallelization of the Breadth First Search (BFS) algorithm to reduce the searching time for a vertices and paths. The BFS explores the edges of a graph to find all vertices reachable from the

root and produces an array that gives the level of vertex. Gayatri and Baruah (2008) stored the adjacent vertices of a given vertex all of which are inserted in the list of vertices that must be visited at the current level. Then, theses vertices are distributed among processors reducing in that the inner processor communication and the latency in finding the requested vertex. Bonacic *et al.* (2008) used the parallelism to reduce the time to find the top R-results for a user query in a web search engines.

Web search engine works as follows, after the user issue a request, two operations fetching and ranking are done to find the user's request. The ranking phase takes more resources when the number of user and data are increased the resources available become insufficient. For that it uses a hybrid parallelization mechanism that is based on a mice MPI (BSP) open MP programming model in which the selection of top-R results for a query is done in parallel using open MP threads, reducing in that the total running time under high traffic scenarios.

That is done by putting T threads to research on the document ranking phase of a group of queries being processed altogether at the same mode in a given period of time with $Q \geq T$ (this is done in parallel). The results show that the speed up achieved in the heavy ranking phase for two nodes with higher or moderate query traffic increases with the number of threads. In addition, it is possible to achieve efficiencies near to 85% in a particular system with 32 nodes. Marin *et al.* (2003) compared different parallelism mechanism for searching in suffix array.

Suffix array are based on binary searching given a text collection, the suffix array contains a pointers to the initial positions of all retrieval strings. For example, all the word beginnings are used to retrieve words and phases or all the text characters are used to retrieve any substring. These pointers identify both documents and positions within them. Each such pointer represents a suffix which is the string from that position to the end of the text. The array is sorted in lexicographical order by suffixes. The efficiency of parallelization of suffix array is involved because its element points to random text position that covers the whole collection. The obvious way is to produce an independent suffix arrays in each processor and broadcasting every query to all processors.

Marin *et al.* (2003) compared the running time of the sequential suffix array, local suffix array where suffix array constructed in every processor and lexico suffix array that is global array partitioned in p pieces and distributed on different processors in order. Vlexico is similar to lexico but the distribution of pieces is done in circular manner among processors.

Global is similar to local but the search in this strategy continues in the other processors if a given query is not found. The results show that global performs are better than the other algorithms for four processors. However, local is the worst one in the running time but it is very efficient in communicating hardware compared to the others. Digalakis and Margaritis (2003) used the parallelism to reduce the running time of serial evolutionary algorithms. Evolutionary algorithm is a powerful tool used to find near optimal solution of complex problems within a relatively little amount of time compared to the problem's very large search space. EA follows the rules of nature that fittest individuals of current population survive and are chosen to reproduce the next generation.

There is a little chance that some individuals with small fitness will be chosen for reproduction and that mutation may take place to provide diversity to population. These processes keep going on until the solution is found or the criterions are met. The use of these techniques results in a population with higher average fitness for each generation.

EA has some problems for a huge amount of computation time. For that parallelism is introduced to solve this kind of problems. Since, fitness evolution is inherently independent, the most intuitive implementation of parallelism is to divide the population into several chunck of equal size and distribute each of them to every processor who calculates the fitness of each individual in the assigned chunk. As each processor performs selection and crossover operation independently on its own population and only exchanges individuals occasionally the communication overhead is lower. The results show that for a large population size and long evolution time, the speed up is increased linearly with number of processors.

SEQUENTIAL PARALLEL SEARCH ALGORITHM (SPSA)

Algorithm description: The research is based on reducing the time needed to search an item in unordered array located in any location. Searching unordered array saves the time needed for sorting and reduces the number of comparisons needed.

SPSA works as follow: For a given input data A with size n, the array A is divided into two subarrays A_1, A_2 each with size n/2. If n is odd number the array is divided into subarrays of size n/2, (n/2)+1. To each subarray a processor P_1 is assigned to A_1 and P_2 is assigned to A_2. The two processors start searching the requested item in parallel from the left. When one of the two processors

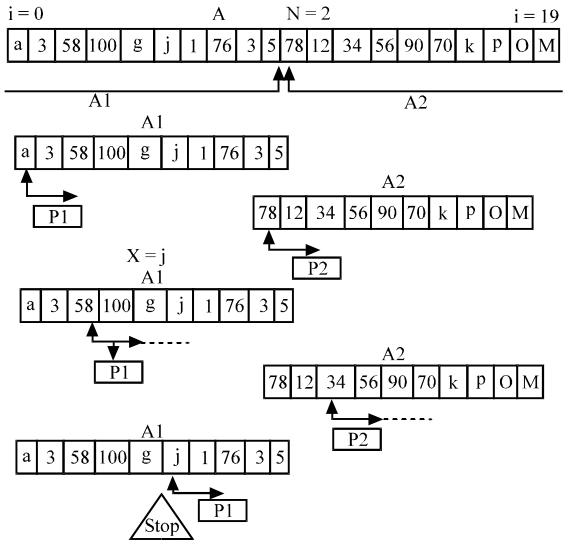

Fig. 1: Example of searching machanism in SPSA

Table 1: Running time comparison of an array of size n

Comparison	SPSA running time	Binary search running time
Sorting	-	Vary from n^2 to n log n
Array division	Log n	Log n
Searching in the worst case	n/2	Log n
Total running time	n/2+log n	Vary from n^2+2log n to (n+2) log n

finds the requested item, it sends a message to the second processor and the searching operation is stopped. In the research we can find the following cases:

- If the requested element is not found in A (worst case), the two processors scan all the n/2 items and the searching time in this case is O (n/2). The number of comparison is n
- If the item is repeated in the array, the first processor that finds it, it informs the other about that and the searching process is ended
- The best case run time of the algorithm is O (1)

The algorithm has a good performance in term of insertion because the new item can be inserted in any location and the searching time complexity does not change. SPSA outperforms the binary search algorithm in searching an item for the first time where binary search sorts the array before searching. Despite, the binary search has optimal running time in searching sorted array where the searching space is reduced to half in each step until finding the requested item. The algorithm has a fixed performance for searching and insertion. The worst case running time in binary search varies from (n^2+log n) to (n+1) log n that depending on the sorting algorithm used.

For large number of n, this complexity is high. However, the running time of SPSA in the worst case is fixed and is O (n/2) that is much lower than binary search.

Example: Figure 1 shows an example of an array A of size n = 20. SPSA divide A into two subarrays A_1, A_2, P_1 is assigned to A_1 and start searching from the left from i = 0 to n/2. P_2 starts searching A_2 from i = (n/2)+1 to 19. The searching item is j.

Complexity analysis: Table 1 shows a comparison between the running time of the algorithm SPSA and the running time of binary search algorithm. It is clear that the running time of SPSA is the better than the running time of binary search algorithm in the worst case.

SIMULATION RESULTS

SPSA is implemented using Visual Basic 6 on stand-alone PC with Pentium III processor running at 800 MHz under windows XP platform. The parallelism is simulated using timer controls for the representation of multiprocessors environment where each control represents a single processor. On the same infrastructure the binary search algorithm is implemented and a comparison of the running time of both algorithms is done taking in consideration the following points:

- In SPSA is calculated the time needed for splitting and searching separately and the total running time is the sum of these 2 times
- In binary search is considered the time of sorting and searching separately and the running time is the sum of these both times (the sorting algorithm used is Quicksort)

In the research two cases are considered: true case where the item is in the array and the false case where the item is not in the array. From Table 2 and 3 it is clear that the searching time increases linearly with the size of items in both cases. Comparing the results shown in Table 2 and 3 researchers can conclude that SPSA outperforms the performance of binary search.

Performance analysis: In parallel systems, two parameters are used to study the efficiency of parallel systems: the speed-up that indicates the factor by which the execution time for the application change and is calculated as follow:

$$\text{Speed-up (Accelerator)} = \frac{\text{Execution time for one processor}}{\text{Execution time for P processors}}$$

Table 2: Running time of SPAS and binary search in true case

No. of items	SPSA running time (m sec)	Binary search running time (m sec)	Speed-up
5000	18.52500	69.30800	3.740
50,000	66.40500	554.6860	8.350
500,000	238.2810	6.085937	25.54
1,000,000	480.5390	12.906319	26.80
5,000,000	2.550781	71.390261	27.98

Table 3: The running time of SPSA and binary search in the false case

No. of items	SPSA running) time (m sec)	Binary search running time (m sec)
5000	19.53000	109.37300
50,000	78.12400	570.31100
500,000	250.2810	6.3479370
1,000,000	492.2560	12.918036
5,000,000	2.835935	71.675775

Table 4: The efficiency or SPSA

No. of items	Accelerator	Efficiency (%)
5000	3.740	1.870
50,000	8.350	4.170
500,000	25.54	12.77
1,000,000	26.80	13.40
5,000,000	27.98	13.99

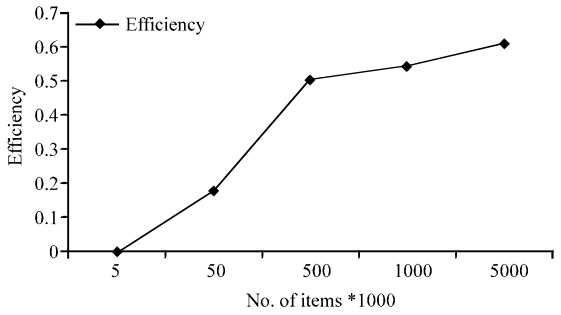

Fig. 2: The efficiency fo SPSA over binary search algorithm

Table 2 shows the speed up of the system in the true case. The speed up increases linearly with the number of items but the increasing ratio is high for small input size. Binary search performs better when the input data is increased. The second parameter used in evaluating the performance of the parallel systems is the efficiency that is the accelerator/ No. of processors %. Table 4 shows the efficiency of the system considering that the number of processors used is 2.

Comparing Table 4 and Fig. 2, we can note that the efficiency of the system over the binary search increases linearly when the number of item increases. But for a large input size the variation in efficiency is low for (1,000,000, 5,000,000 item) and that because the binary search performs better for large input size. Given that the system is scalable because its performance is not degraded with the increasing in the input size and it gives a good efficiency.

CONCLUSION

In this study researchers propose a parallel searching mechanism SPSA in which the idea is to eliminate the need of sorting the input array in order to search an item. The proposed algorithm works in unsorted array using two processors that is search the two parts of the array in parallel. To study the effectiveness of the system researchers investigated and compared the running time of the proposed algorithm SPSA with that of binary search by conducting a series of simulation experiments data to evaluate the performance of SPSA. The experiments results confirm that the efficiency of the proposed algorithm increases and the time needed to search an item is reduced up to 26% for an input size of 5000-1,000,000 items and up to 3.6% for large input size.

The results shown in Table 2 show that the performance of the algorithm is better than the binary search because we have eliminate the sorting time which is required for the binary search. In addition, the algorithm has fixed time complexity in the worst case. SPSA algorithm outperforms the binary search in two aspects: first having an odd input size does not influence in the time complexity.

However, in binary search this case persists for each splitting operation. The second aspect is that inserting a new item in the array in the algorithm an item can be inserted in any location (head, tail, middle) and that does not influence the searching time. Beside that the item can be found in the array with same time complexity. Nevertheless, in sorted array which is the binary search algorithm required, the item must be inserted in the correct location to be found later by binary searching algorithm.

REFERENCES

Bonacic, E., C. Garcia, M. Marin, M. Prieto and F. Tirado, 2008. Exploiting hybrid parallelism in web search engine. Proc. Int. Euro-Par Conf. Parallel Process, 5168: 414-423.

Cazenave, T. and N. Jouandeau, 2007. On the parallelization of UCT. Proceedings of the Computer Games Workshop, (CGW'07), University of Maastricht, The Netherlands, pp: 185-192.

Chaslot, G.M., M.H. Winands and H.J. Herik, 2008. Parallel monte carlo tree search. Proceedings of the 6th International Conference on Computer and Games, Sept. 29-Oct. 1, Springer-Verlag, Beijine, China, pp: 25-36.

Cormen, T., C. Leiserson, R. Rivest and C. Stein, 2001. Introduction to Algorithms. 2nd Edn., McGraw-Hill Book Company, Cambridge and New York.

Coulom, R., 2007. Efficient selectivity and backup operations in monte carlo tree search. Proc. Int. Conf. Comput. Games, 4630: 72-83.

Digalakis, J. and K. Margaritis, 2003. Parallel evolutionary algorithms on message passing clusters. http://www.it.uom.gr/people/digalakis/digamarg2003.pdf

Gayatri, R.K. and P.K. Baruah, 2008. Parallelizing breadth first search using cell broadband engine. http://www.hipc.org/hipc2008/documents/HiPC-SS08-FinalPapers/1569154967.pdf

Knuth, D., 1997. The Art of Computer Programming Sorting and Searching. 3rd Edn., Vol. 3, Addison Wesley, USA.

Kocsis, L. and C. Szepesvari, 2006. Bandit based monte-carlo planning. Mach. Learn.: ECML., 4212: 282-293.

Koorangi, M. and K. Zamanifar, 2007. Designing optimal binary search tree using parallel genetic algorithm. Int. J. Comput. Sci. Network Secur., 7: 138-146.

Marin, M., J. Vega and R. Mirande, 2003. Comparative study of parallel suffix arrays algorithms. Workshop Chileno Sistemas Distribuidos Paralelismo, 2857: 311-325.

Williams, J.W.J., 1964. Algorithm 232: Heapsort. Commun. ACM., 7: 347-348.