Mathematical Problems of Computer Science 35, 14--18, 2011

Implementation of a Search Parallelization Method on High Performance Computing Structures

Grigor H. Geoletsyan and Hrant G. Geoletsyan

Institute for Informatics and Automation Problems of NAS RA
Russian-Armenian (Slavonic) University

Abstract

We describe the implementation of modified initialization and load balancing methods during parallelization of search algorithms for solution of optimization problems on high performance computing structures.

Most optimization problems in discrete mathematics related to applications in new research areas are NP-hard. Parallelization of search algorithms using static and dynamic load-balancing (distribution of computational resources) and their implementation on high performance computing structures, in particular, on computer clusters is an essential tool for obtaining effective solutions to such problems.

Description of search process requires specification of the order of visiting of graph vertices corresponding to states of the search process. Examples of uninformed search methods are the breadth-first search, depth-first search, depth limited, iterative deepening search (IDS). For these methods, the location of targets does not affect the order of vertex disclosure, systematic generation of new states and checking whether they are a solution. Informed search methods, in contrast to the methods of the uninformed search, find solutions more efficiently, using heuristic information specific to this problem - information about the global nature of the graph and the general location of the target search. Examples of informed search methods are A* and IDA* [8]. *A* search method* is an efficient optimal algorithm of this type with the use of this heuristic. That is, no optimal algorithm consider fewer vertices than A* (except for possible consideration by the vertices with the value of heuristic function - the estimated cost of the path to a solution passing through the vertex n - $f(n) = C^*$ in a different sequence, where C* - evaluation of optimal solutions). Despite the fact that the method of A*, in general, performs significantly fewer transactions than uninformed search methods, for most of the tasks it still requires an exponential number of operations. The drawback of this method - a large amount of memory needed to store information about the tops, often does not yield the desired solution, due to limitations on available memory. Proceeding from this application of full and optimal search techniques, saving memory in use at the expense of execution time is prefferable.

Such a method is, for example, *the method of iterative deepening*, which is an effective tool to reduce the demands on memory. By applying this method to A*it is obtained an iteratively deepening A* search method (IDA*) [8], which at each iteration seeks a solution to the estimate f, passing through all vertices with an estimate of less than or equal to f. The time complexity depends on the choice of heuristic function. Reducing the time spent also contributes to the lack of need to store the list of priorities.

Application of high performance computing structures, in particular, the clusters allows parallelizing the process of finding solutions in the state space. Use of effective schemes to parallelize (e.g.

parallelization breadth) enables solutions a wider range of problems that makes it important to develop effective methods for parallelization.

The overall strategy to parallelize the search involves finding the initial distribution of states (vertices of the tree) to processors (*initializing of search*). Each processor performs a local search of solutions, besides processors process incoming requests for work from other processors. Upon completion of the local examining of vertices, a request for allocation of work is initiated. To achieve uniformity of work static and dynamic *load balancing* methods which allow the distribution of vertices during the implementation of search should be used.

In [2] there were proposed two methods to initialize the parallelization of the search - INIT1, INIT2. In the method INIT1 all processors start searching from the root node and perform breadth-first search until the number of vertices in the list of unclimbed vertices becomes less than the number of processors. Each processor performs the same operations and they have identical lists of identified but not viewed vertices. After that, each processor chooses a deterministic way to the top for local search so that all nodes have been distributed and no node is located at the two processors. The advantage of this method is lack of data transfers between processors without increasing the execution time and reducing the acceleration of the parallel algorithm.

Initialization method INIT2 is a static method of balancing. All processors start searching to the width from the root. Then the vertices are distributed among the processors. It should be noted that not all vertices may have the same weight (a value proportional to the number of vertices in the corresponding subtree), since vertices at a lesser depth, on average, have larger (by the number of nodes) subtrees, and therefore require more time to process them.

Uneven distribution of work between the processors can cause inefficiency of the method to parallelize. Using dynamic load balancing method allows achieving uniformity of distribution. Since under the initial partition of the tree it is difficult to predict the size of the subtree of the top, different methods are used for dynamic load distribution: asynchronous carousel, selection a random neighbor, and so on. A distinctive feature of the latter is that the distribution of work is going on locally. This leads to the fact that the queries are repeated often. This can be avoided by using the global distribution of work, which provides a better distribution. To find the optimal solution the parallelization of the search is conducted on equal depths of the search tree. For this purpose, each processor periodically requests the donor processor to provide vertices situated on the upper tiers of the search tree. Based on the information on the levels of processed vertices received by the requesting processor, the donor processor takes an appropriate decision.

Upon receiving a request for the task the processor - donor passes a part of the work, the amount of which is approximately equal to half the total count of vertices in the list. This strategy helps to ensure approximately equal proportions of promising and less promising vertices in the list of processors which improves the efficiency of the parallel algorithm.

For iteratively deepening informed depth-first search IDA* in the division of work it is proposed to consider not the depth of the vertex, but the value of the estimation function at the given vertex. For division of works the vertices in the stack are ranged in ascending order of values of the evaluation function, in case of the equal values of the estimate the depths of the vertices are taken into account: the vertices located deeper are located higher in the sorted list. This process of a modified depth-first search is repeated at each iteration IDA* for as long as the solution is found, or the search has been stopped.
If finding of any solution is sufficient, then the processor that has found the solution stops the operation of other processors. If you can not implement this mechanism of work quitting in a given architecture, the processors must periodically inquire each other on the indicator of a solution. If you want to find all solutions (or all of the best solutions) of the problem, then the parallel depth-first search views some portion of the search space (defined by the algorithm and heuristics used). Since the search space is dynamically shared between processors, to verify the completion of work by any processor is not sufficient to query processors in a successive order. The reason is that the processor can get the job after having been interrogated. Therefore using methods of distributed assessment of the work completion of processors is required that can be done, for example, using the Dijkstra's algorithm [6].

Thus, when parallelizing IDA* search strategy initialization INIT2 and method of distribution of work during the dynamic load balancing [2], [3] are proposed, which can be applied to a deep branch and

bound method (DFBB) with some modifications, in this case the processors must know the best solution currently available.

For architectures with shared memory the condition is easily achieved by storing the best solutions in the shared memory, and for computers with message transmission, each processor stores the best known solution. Once a processor finds a better solution, it shall inform all other processors, which, if necessary, update the local best solutions. Note that if the best solution known to this processor is worse than the global best solution, it can affect only the efficiency but not on the correctness of the algorithm. Additional time costs associated with keeping the best solution make a small fraction of expenditures required for the dynamic balancing of loads.

Developed methods to parallelize IDA* - initialization, dynamic balancing have been applied to problems solved by systematic search, for example, the problem of finding disjoint paths between k pairs of vertices in a directed graph and bounded vertex cover of a bipartite graph [1].

The task of finding a bounded vertex cover has many important applications, especially in the production of high-performance VLSI, since, with increasing integration of chip production, the problem of increasing the efficiency of production arises. Because of this, it is important to use methods of fault tolerance for an acceptable level of efficiency in output [4], for example, in the manufacture of circuits containing very simple structure - the memory. This is due to the increasing demand for storage capacity and the continuous percent growth area of memory from total area of the chip. One of the methods adopted to increase the useful output in the production of memory is to add redundant rows and columns, which can be used in carrying out internal or external reconfiguration, during which the faulty columns and rows are replaced by the reserve. Kuo and Fuchs [7] showed that the problem belongs to the class NP-complete problems and conducted a study of the location problem of redundant elements used in reconfigurable VLSI memory. This problem is formalized as a problem of restricted vertex cover of a bipartite graph where one subset of vertices of a bipartite graph corresponds to the rows, and the other subset to the columns of a rectangular array.

The task of finding a bounded vertex cover of a bipartite graph [7] has the following form.

Given are a bipartite graph $G(V_1, V_2, E)$ and two positive numbers k_1 and k_2. Find subset $C_1 \subseteq V_1$ and subset $C_2 \subseteq V_2$ such that for every edge in E at least one of its vertices belongs to $C_1 \cup C_2$ and the amount of $|C_1| + |C_2|$ is minimal.

To obtain high performance programs, we chose a data structure for the effective implementation of basic operations. The data structure of the graph is stored globally and during the search it is not changed, only variables of vertices on the state tree, referring to given covering change.

The top of the search tree is characterized by subsets of $C_1(n)$. $C_2(n)$ sets $V_1(n)$. $V_2(n)$. which correspond to an intermediate vertex cover. Component $g(n)$ of heuristic function $f(n)=g(n)+h(n)$ [1] in this case is equal to $|C_1(n)| + |C_2(n)|$. Component $h(n)$ is calculated as follows. For all values of x from 0 to $k_1-|C_1|$ x uncovered vertices (ie, not belonging to the covering C_1) from the set V_1, with the largest number of uncovered incident edges are chosen. The number of y vertices of V_2, you need to cover, is calculated in such a way as to guarantee the admissibility of the heuristic function. Compared with the heuristic function for searching for the solution of this problem by IDA*, given in [8], the functions described above can permit considering a smaller number of vertices (in some cases up to two times). The results of testing the performance of a parallel program using the heuristic function and dynamic balancing confirm that the number of options considered in the state space is significantly reduced and a high degree of parallelization is achieved [2], [3]. It should be noted that the greatest effect of the proposed methods for parallelization is shown for large values of k_1 and k_2. In this case, however, keep in mind that an increase in k_1 and k_2 is limited with computing capacities, due to the exponential nature of the algorithm.

In software implementation of the proposed approach a parallel library of Parallel Boost Graph Library (PBGL), installed and tested in a parallel cluster environment "Armcluster" within the target project "Creation of the state system of scientific computing Armenia" is used.

To test the proposed approach (define the run-time of the parallel algorithm) in [10] examples, consisting of 14 classes, which differ by their characteristics, were used. For each class of instances 100 random tests were generated. Figure 1 shows comparative temporal characteristics of some algorithms given in [10], the method uses an iterative deepening search (IDS), and iteratively deepening of the modified method of informed search (IDA*) using the proposed methods of initialization and balancing.

The work of algorithms stopped, if the duration of execution exceeded 100 seconds. Temporal characteristics of the modified IDA* have confirmed its acceptability in the solution of the problem (by 13 classes of examples of the proposed modification has better time indices).

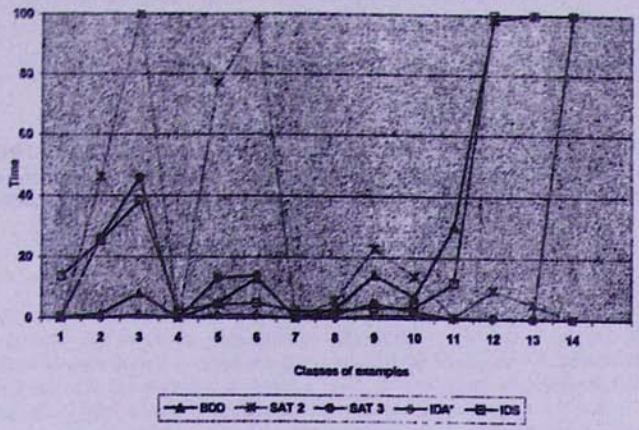

Fig.1. Comparative time performance of algorithms.

References

[1] Г. Геолецян, "Эвристический алгоритм решения задачи ограниченного вершинного покрытия двудольного графа", Доклады НАН РА, т. 107, н. 1, стр. 44–48, 2007.

[2] Г. Геолецян, Г. Геолецян, "Инициализация параллельного поиска", *Годичная научная конференция РАУ. 2007. Сборник научных статей. Ереван*, стр. 115-121, 2008.

[3] Г. Геолецян , Г. Геолецян, "Инструментальные средства реализации методов параллельного поиска" Вестник РАУ. Ереван. 1, сс. 32-40, 2008.

[4] I. Kim, Y. Zorian, G. Komoriya et al., "Built in self repair for embedded high density SRAM", *Proc. Int. Test Conference*, pp. 1112–1119, 1998.

[5] H. Fernau, R. Niedermeier, "An efficient exact algorithm for constraint bipartite vertex cover", *J. Algorithms*, vol. 38, no. 2, pp. 374–410, 2001.

[6] A. Grama, V. Kumar, "State of the art in parallel search techniques for discrete optimization problems", *IEEE Trans. Knowl. Data Eng.*, vol. 11, no. 1, pp. 28–35, 1999.

[7] S.-Y. Kuo, W. K. Fuchs, "Efficient spare allocation in reconfigurable arrays", *DAC*, pp. 385–390, 1986.

[8] M. G. Lagoudakis, "An IDA* Algorithm for optimal spare allocation", *SAC*, pp. 486–488, 1999.

[9] N. R. Mahapatra, S. Dutt, "Random seeking: A general, efficient, and informed randomized scheme for dynamic load balancing", *Int. J. Found. Comput. Sci.*, vol. 11, no. 2, pp. 231–246, 2000.

[10] F. Yu, C-H. Tsai, Y-W.Hang, H-Y. Lin, D. T. Lee, S-Y. Kuo, "Efficient exact spare allocation via boolean satisfiability", *Proceedings of the 20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'05)*, Oct., Monterey, California, 9p, 2005.

Հատարկման զուգահեռացման մեթոդի իրականացումը բարձր արտադրողականությամբ հաշվողական կառուցվածքներում

Գ. Գեղեցյան և Հ. Գեղեցյան

Ամփոփում

Նկարագրված է համակարգված հատարկման ալգորիթմների զուգահեռացման սկզբունքների և բալանսավորման մոդիֆիկացված մեթոդի իրականացումը բարձր արտադրողականությամբ հաշվողական կառուցվածքների վրա օպտիմալացման խնդիրների լուծման ժամանակ: