Priority of Tasks and Performances of Branch-and-Bound Load Balancing Strategies

Salah DOWAJI and Catherine ROUCAIROL
Université de Versailles-St Quentin - Laboratoire PRiSM
45, avenue des Etats-Unis
78000 Versailles FRANCE
e-mail: {sad,roucairo}@prism.uvsq.fr

1 Introduction

Branch and Bound (B&B) algorithms are the most popular techniques used to solve NP-hard combinatorial optimization problems. With this kind of problems, too many tasks/subproblems (represented as nodes in the B&B tree) are created dynamically so that load balancing becomes an important factor to optimize performance. In such cases, subproblems are classified in function of their priorities into several queues, and solutions may be reached more efficiently if priorities given to subproblems by queues are taken into consideration during the parallel execution of the algorithm.

Existing load balancing schemes use partial cost to balance subproblems generated by best-first B&B distributed algorithms.

In this article, we show that using partial cost might lead to a big accumulation of high priority tasks on a few processors, while the other processors continue to work with low priority subproblems. In such a situation, an other notion of priority is needed. It must take into account not only the partial cost of subproblems but also their capacity to generate other subproblems.

Our new notion of priority is applicable to various applications of which those using best-first B&B algorithms and those from the area of Operational Research. We chose to evaluate the strategy with a B&B solution of the Vertex Cover Problem (VCP).

Our aim with using the B&B algorithm is not to study the parallelization of this method, but rather to develop an efficient load strategy that uses priority of tasks.

The implementation of the VCP was carried out in a network of heterogeneous Unix workstations used as a single parallel computer through a Parallel Virtual Machine (PVM) software system. This system with its communication facilities provides the flexibility to develop parallel applications [1,2]. Experiments in this article have been done on 3 "SUN" workstations (1 SPARCstation ELC and 2 SPARCstations IPC).

We have developed three load balancing strategies: a "one-by-one" and two versions of "partial distribution" strategies.

In this paper, we start by describing the parallelization of B&B and the implementation of the VCP under PVM. Afterward, we discuss the motivations and justifications to find an other notion of priority between different subproblems. Finally we present the advantages of our new notion of priority by showing the improvement in performance of the developed load balancing strategies. As a conclusion, we point out that our approach can be applied to a lot of problems based on tree search space exploration.

2 The Branch & Bound VCP implementation

Branch & Bound [3] is based on exploration of search space to find the optimal solution, but it cuts off parts of this space by using heuristic knowledge about the position of the solution in it. Nevertheless, the size of the search space still grows exponentially with the size of the input. Thus, it is until now an open research problem to study, and the main question which we try to answer with many other people [5,7,8,9], is what can parallelism offer in solving NP-hard combinatorial optimization problems.

The B&B algorithms use a large class of search techniques. Among them, there is the best-first B&B. This technique handles the subproblem with minimal bound. It leads to a very good computational performance but it is very expensive in memory utilization [3,6,7].

In this paper, we use a distributed version of best-fist B&B to solve the Vertex Cover Problem (VCP). For an undirected graph $G = (V, E)$ a subset $U \subseteq V$ of nodes is called a vertex cover if every edge is covered by a node from U, i.e. $\{u, v\} \in E$ implies $u \in U$ or

$v \in U$. This problem aims to find the vertex cover with minimal cardinality and it is known to be NP-complete [5,6].

All subproblems are coded as tuples (SC, pc) where SC is a sub-cover (a set of nodes that cover some edges of the graph G) and pc is the number of nodes in SC; we call it from now on the **partial cost** of the treated subproblem (lower bound on the sub-cover cardinality).

Our B&B algorithm generates two new subproblems (SC_1, pc_1), (SC_2, pc_2) from a given subproblem (SC, pc). The generation of SC_1 and SC_2 by the branch procedure is done by searching for a node v with maximal degree in the rest-graph (part of the original graph that has not yet been covered). $SC_1 = SC \cup \{v\}$, $SC_2 = SC \cup \{w \in E\}$ (for more details, see [5]).

We have implemented the distributed B&B scheme with different strategies of load balancing as we will see later. Our aim in using the B&B algorithm is not to study the parallelization of this method, but rather to develop an efficient load balancing strategy that uses priority of tasks, so that the chosen algorithm is sufficient for our purpose (for more information about B&B algorithms and their parallelizations, refer to [3,7]).

The chosen B&B algorithm uses a distributed queue organization to temporarily stock the generated subproblems. We chose to execute only one process per processor in the PVM machine. Each processor has a local queue in its local memory and executes the sequential B&B algorithm using this queue.

In our implementation, we have used the best-first technique. Subproblems are classified in many queues in decreasing order of their priority. The evaluation of a subproblem is given by its partial cost.

With the distributed B&B scheme, the communication plays an essential role in determining the performance especially when we use multi-user machines (like workstations). To improve performance, we must try to reduce the number of exchanged messages during the execution, as we will show in the following.

3 Studied load balancing strategies

We have developed three load balancing strategies, a "one-by-one" and two versions of "partial distribution", to understand load distribution between the different slaves. We started by testing all of these strategies with the generated subproblems of each processor classified into a local queue according to their associated **partial cost**.

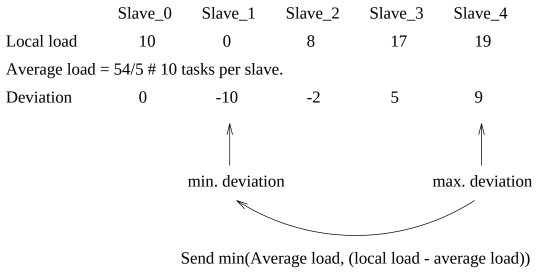

Figure 1: An example of application of partial distribution load balancing strategy, where slave_1 asks the other slaves for work.

In the one-by-one load balancing strategy, each slave needing work receives only one subproblem from the beginning of the queue of an other loaded processor.

In the partial distribution strategies, the number of tasks received by a processor requiring work is variable. The number of subproblems sent from the maximum loaded slave to the one requiring work is equal to the minimum of the average load and the local load minus the average load (see the following example given by figure 1).

The first version of partial distribution load balancing strategy performs load balancing by sending subproblems from the beginning of the local queue whereas the second one sends subproblems from the end of the queue.

We have noticed that we get a very bad distribution of load in the case of the "one-by-one" strategy; at the beginning all slaves have the same quantity of work but after that one of them becomes strongly loaded although this slave is not the one where the root subproblem in the B&B tree has been affected.

We can explain this situation by the fact that some subproblems potentially generate more work than the others in the search space.

We have noticed also that with version 2 of the partial distribution load balancing strategy, the loads of different slaves are globally more equally distributed comparing to the version 1, and that with version 2 load balancing operations occur less frequently than with version 1. This proves that the **partial cost** used as a criterion to classify subproblems within the queues is not the only parameter to apply to get a good load balancing strategy.

The obtained results indicate that it is important to take into consideration the potential work that subproblems can generate.

Any efficient load balancing strategy should ensure, as far as possible, that the processing of subproblems occurs in the global order of their partial cost and should take into consideration their associated potential work. Consequently, we need to define a priority notion between different subproblems which takes into consideration these two parameters.

4 A new notion of priority: potentiality

A generated subproblem by an execution of B&B algorithm has two associated values: its cost and an indicator representing the potential work it can generate successively to reach an optimal solution (see figure 2). These two values depend on the treated problem. In the VCP problem for example, the first one is equal to the partial cost, and the other value will be taken equal to the number of edges in the rest-graph.

Each generated subproblem is characterized by its position into the indicated region in figure 2. The execution of B&B algorithm starts from **Start** point (where we have the root of B&B tree) and terminates at **End** point (the searched optimal solution).

The potentiality of a subproblem sp is defined as:

Definition .1

$$Potentiality(sp) = (dist(sp, End),\; potential\_work,\; ancestors\_nb).$$

Where:

sp is the considered subproblem,

$dist(sp, End)$, the distance between the position of sp and the **End** point,

$potential\_work$, the value of the associated potential work of sp,

$ancestors\_nb$, the number of the ancestors of sp.

In this definition of priority, we introduced a third parameter, the number of ancestors, to select between subproblems that have the same distance from the End point and the same quantity of potential work. We chose to privilege those having the less number of ancestors.

In practice, the value of the optimal solution is not known before the end of the execution, but usually we start execution with an upper bound that maximizes this value.

In the VCP problem, for example, we can use the number of nodes of a given graph as an End point and we compute distances relatively to it.

Generated subproblems are classified into the different queues during the execution of B&B distributed algorithm as a function of their priorities. A subproblem sp_1 has more priority than another subproblem sp_2, if it verifies one of the following rules:

$$dist(sp_1, End) < dist(sp_2, End) \tag{1}$$

$$(dist(sp_1, End) == dist(sp_2, End))\&\& \\ (sp_1.potential\_work > sp_2.potential\_work) \tag{2}$$

$$(dist(sp_1, End) == dist(sp_2, End))\&\& \\ (sp_1.potential\_work == sp_2.potential\_work)\&\& \\ (sp_1.ancestors\_nb < sp_2.ancestors\_nb) \tag{3}$$

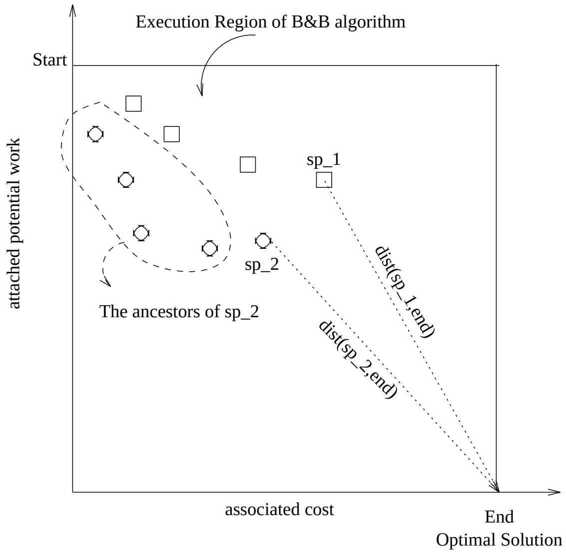

Figure 2: Definition of potentiality of two subproblems sp_1 and sp_2.

5 Influence on load distribution

For the "one-by-one" load balancing strategy, we have remarked that at the beginning of the execution, the assigned subproblems have more priority than the others so that the number of load balancing operations is lower than its number without using our priority notion. At the end, we have a lot of load balancing operations when the priority of subproblems begins to decrease.

The load is equally distributed with partial distribution load balancing strategies (version 1 and 2).

We have also noticed that the load of the most loaded slave does not exceed the 110 subproblems while it was around 450 subproblems (with version 1) and 600 subproblems (with version 2), without using our potentiality notion.

6 Influence on communication costs

To show the importance of the communication factor, we have chosen to study the exchanged messages between the master and all of its slaves.

For the "one-by-one" strategy, we have remarked that there is a clear difference in the number of sent and received messages with and without using potentiality notion. This number is lesser in the first case.

For the two versions of partial distribution load balancing strategy, we have remarked that the number of sent or received messages takes an asymptotic value. In version 1, the number of exchanged messages remains approximately the same, while in version 2 the utilization of our priority notion lightly increases the number of exchanged messages. In fact, with this version, when an unloaded slave takes subproblems from the end of the queue of a loaded slave, this unloaded slave executes the work and rapidly becomes unloaded. This light augmentation of the number of load balancing operations shows that subproblems are really classified in queues as a function of their priority of execution.

7 Conclusions & perspectives

We analyzed the behavior of three load balancing strategies and proved the necessity to have a new priority order between the different executed subproblems in a B&B application.

We introduced the notion of potentiality based on an estimation of the capacity of a subproblem to generate work, and on an evaluation of the distance between the subproblem and the optimal solution. We believe that this kind of potentiality can be applied to any B&B application where subproblems are characterized by the previous parameters. The use of our notion of potentiality with other B&B applications will be explained during the presentation of this paper.

Furthermore, we studied the impact of this notion on the behavior of load balancing strategies for distributed best-first B&B applications. These strategies show that after taking into consideration the potentiality of subproblems, their selection to participate to an operation of load balancing is not the same at the beginning and at the end of the execution.

All these load balancing strategies are developed in a centralized manner (there is one master processor that controls the other ones in the PVM machine). Future works will consist to investigate the effect of the use of our potentiality notion when several masters instead of only one (like the strategies in [8,9]) are used.

References

[1] Al Geist, Adam Beguelin, Weicheng Jiang, Robert Manchek and Vaidy Sunderam, "PVM 3 User's Guide and Reference Manual," Oak Ridge, Tennessee 37831, May 1993.

[2] V. S. Sunderam, G. A. Geist, J. Dongarra and R. Manchek, "The PVM Concurrent Computing System: Evolution, Experiences, and Trends," *Parallel Computing*, Vol. 20, No. 4, pp. 531-546, March 1994.

[3] C. Roucairol, "Parallel Computing and Combinatorial Optimization," EURO XIII, Operational Research Designing Practical Solutions, pp. 229-254, July 1994.

[4] Ten-Hwang Lai and Rartaj Sahni, "Anomalies in Parallel Branch-and-Bound Algorithms," *Communication of the ACM*, Vol. 27, No. 6, pp. 594-602, June 1984.

[5] R. Lüling and B. Monien, "Two Strategies for Solving the Vertex Cover Problem on a Transputer Network," *In proc. of the 3rd Int. Workshop on Distributed Algorithms*, number 392 in Lecture Notes of Computer Sciences, pp. 160-171, 1989.

[6] R. Lüling and B. Monien, "Load Balancing for Distributed Branch and Bound Algorithms," *Proc. of Int. Parallel Processing Symposium (IPPS)*, pp. 543-549, 1992.

[7] Myung K. Yang and Chita R. Das, "Evaluation of a Parallel Branch-and-Bound Algorithm on a Class of Multiprocessors," *IEEE Transactions on parallel and distributed Systems*, Vol. 5, NO. 1, pp. 74-86, January 1994.

[8] A. B. Sinha and L. V. Kale, "A load balancing strategy for prioritized execution of tasks," *In Workshop on Dynamic Object Placement and Load Balancing in Parallel and Distributed Systems (in co-operation with ECOOP '92)*, June 1992.

[9] Vikram A. Saletore and Mannan A. Mohammed, "Hierarchical load balancing schemes for branch-and-bound computations on distributed memory machines," *In Hawaii International Conference on System Software*, January 1993.