Mathematical Modelling and Analysis 2005. Pages 165–170
Proceedings of the 10th International Conference MMA2005&CMAM2, Trakai
 ISBN 9986-05-924-0

PARALLEL BRANCH AND BOUND ALGORITHM TEMPLATE

M. BARAVYKAITĖ

Vilnius Gediminas Technical University

Saulėtekio al. 11, LT-10223 Vilnius, Lithuania

E-mail: mmb@fm.vtu.lt

Abstract. In this work the ideas of template programming are implemented for sequential and parallel branch and bound (BB) algorithms in the parallel BB algorithm template. Some results of calculation experiments of programs generated using this template are given. The efficiency of implemented parallel algorithms is discussed.

Key words: template programming, parallel branch and bound algorithms

1. Introduction

Parallel optimization algorithms are complicated to implement. For some algorithms, parallel program can be obtained from the sequential one using some parallelization tools. One type of such tools is parallel algorithm templates [1]. Branch and bound (BB) algorithm is suitable for template programming.

2. Algorithm Templates

The general idea of a template programming is to implement a class of algorithms that solve different problems but have the same control structure. For this purpose the problem dependent part of the algorithm must be separated from the general structure of the algorithm. The reusable implementation of latter then can be used to solve different problems [7].

The ideas of template programming can be used for parallel template programming. Then the template must specify the main features of parallel programming: partitioning, communication, agglomeration and mapping [2].

3. Branch and Bound Algorithm

Branch and bound (BB) algorithm is suitable for template programming since it has a strict control structure that can be used for different problems.

Here the following minimization problem is considered:

$$f^* = \min_{x \in S} f(x),$$

where S is the set of feasible solutions, $f(x)$ is objective function which depends on variables $x = (x_1, x_2, \ldots, x_n)$.

First, initial approximation of the optimal value must be calculated by some heuristic algorithm, otherwise it is set to be infinity. Then the solution space is subsequently partitioned into smaller subsets S_j. This algorithm step is called *branching*. For every subspace S_j a lower bound LB_j of the minimum objective function value is calculated. This step of the BB algorithm is called *bounding*. If the bound is larger than the currently known best solution, then the subset S_j can be eliminated from further search, since this subset does not contain the optimum solution. If the obtained bound is lower than the best known value, then the branching step is performed over this subset. A combination of both two steps over the subset is called a *task*. All unexplored subsets are kept in a list.

During each iteration of the BB algorithm one task is processed. The iteration has three main components: selection of the node to process, bound calculation, branching.

The rule how to choose the next subset to examine can be defined in many ways. This makes many variants of the branch and bound algorithm. The most popular rules are: the *best first search* rule where the next task to explore is the one with the lowest bound; the *breadth first search* rule where the oldest task is explored next and the *depth first search* rule where the youngest task is explored [6]. Node selection rules influence the efficiency of BB algorithm and the number of nodes kept in a list. For particular problems some rules can considerably improve the performance rate of the algorithm.

4. BB Algorithm Template

The parallel BB algorithm template proposes C++ classes for implementation of sequential and parallel BB algorithms. MPI library is used for underlying communications. General structure of the template is given in Figure 1.

`BBAlgorithm` implements various sequential and parallel BB algorithms. The algorithm is performed using `Task`, `Solution` and `SearchOrder` instances. `BBAlgorithm` is implemented by the template. `SearchOrder` defines the rules how to select next task for subsequent partitioning. The most popular rules are already implemented in the tool. Class `Task` should be implemented by the user and it defines the problem to be solved and the branching and bound calculation over the subspace. Class `Solution` implements the solution to be found. For parallel algorithms, `Task` and `Solution` additionally have methods that define how to prepare objects for exchange among processors.

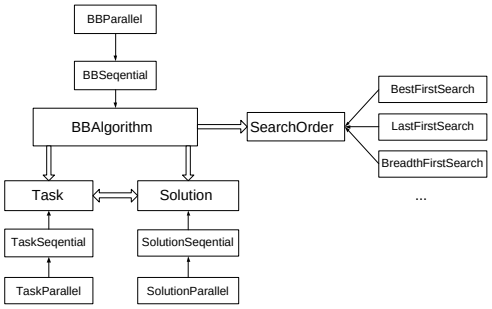

Figure 1. General structure or the BB template.

5. Implementation of Parallel BB Algorithms

Considering partition for parallel execution of BB algorithms, the search performed over any subspace is independent from operations with other subspaces. The domain decomposition algorithm (*DD*) can be used: subspaces of initial search space can be distributed among processors. Additionally, latter generated subspaces are mapped to the same processors and there is no need to remap them. Then BB algorithm can be performed asynchronously over the mapped subspace and at the end the best solution is chosen among them. If the initial search space is divided into more subspaces so that processors get several subspaces the algorithm is called domain decomposition with static balancing (*DD SB*). The domain decomposition algorithm is improved if currently known best solution is exchanged among processors (*DD SE*). When processor finds a better solution it broadcasts it to all other processors and they update their currently known best solutions. The combination of *DD SE* and *DD SB* algorithm is the fourth implemented algorithm (*DD SE SB*).

Parallel optimization algorithms have unpredictably varying unstructured search space [8]. It should be noted that because of the domain decomposition the order of search can differ in parallel and sequential BB algorithm even using the same subset selection rule. Subsets eliminated in the sequential algorithm can be explored in parallel one and it is possible that a total number of the subspaces searched in the parallel algorithm can increase.

Let's define the number of explored tasks as amount of work done by a processor. Then the growth of number of subspaces searched using parallel algorithm can be measured by the *search overhead factor*

$$SOF = \frac{W_p}{W_0},$$

where W_p is the sum of processed tasks in parallel algorithm and W_0 is the number of tasks processed by the sequential algorithm. This coefficient helps to estimate efficiency of the parallel algorithm [3].

Domain decomposition often results in processor load disbalance that reduces the efficiency of parallel algorithms. Communications among processors

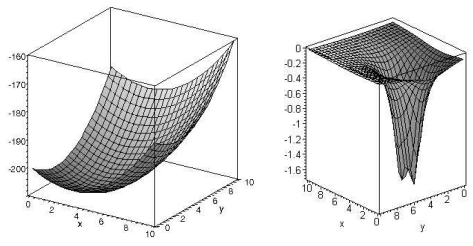

Figure 2. Functions for Lipschitz minimization

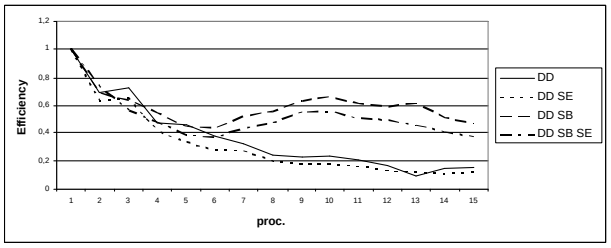

Figure 3. Efficiency of different parallel BB algorithms using the Best first search for the case of Function 1.

introduce additional costs of parallel algorithms. For implemented algorithms communications are not significant compared with performed calculations.

6. Application Examples

The BB algorithm template was used to obtain parallel programs solving Lipschitz function minimization [4] and symmetric traveling salesman problem over 20 cities [5]. Calculations were performed on VGTU cluster Vilkas (www.vilkas.vtu.lt). The examples of optimized Lipschitz functions are given in Figure 2.

For the first function, the efficiency of performance of different parallel BB algorithms is given in Figure 3. Here the *Best first search* rule is used. Algorithms with static balancing performs better because the growth of *SOF*, given in Figure 4, is controlled better.

For other optimized function, all parallel algorithms perform not efficiently (Figure 5). The reason for that is the increase of *SOF* (Figure 6) and great processor load disbalance (Figure 7).

Experiments with traveling salesman problem shows that, parallel BB algorithms performs not efficiently because of growth of SOF and processor load disbalance.

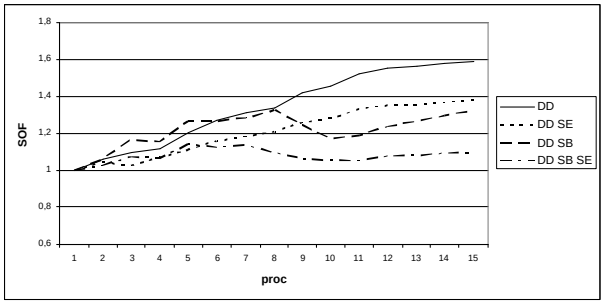

Figure 4. SOF of different parallel BB algorithms using the Best first search (function 1).

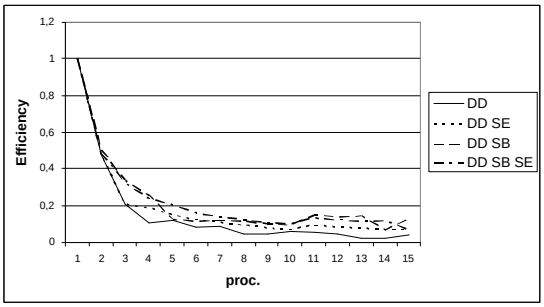

Figure 5. Efficiency of different parallel BB algorithms using the Best first search (function 2).

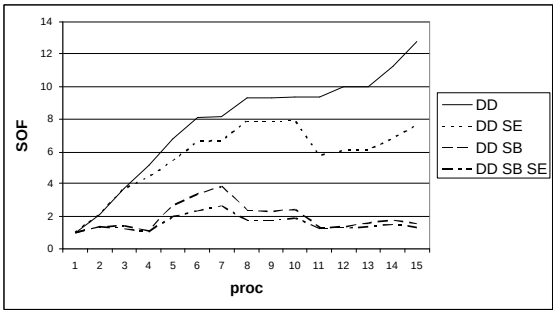

Figure 6. SOF of different parallel BB algorithms using the Best first search (function 2).

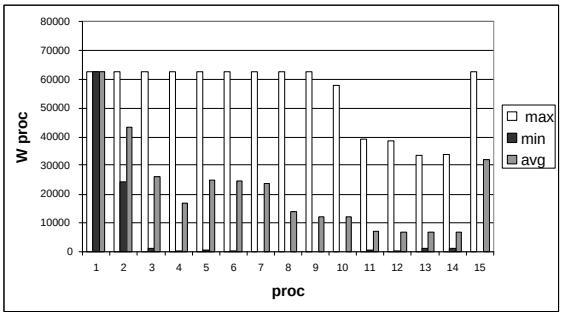

Figure 7. Load disbalance of DD SE SB algorithm (function 2).

7. Conclusions

BB algorithm is suitable for template programming and the BB algorithm template is usable both for sequential and parallel BB algorithm testing. The efficiency of implemented parallel algorithms based on domain decomposition is reduced by growth of SOF and processor load disbalance. SOF can be reduced using solution exchange method combined with static load balancing. Static load balancing balances the processor load not enough for problems solved using parallel branch and bound algorithms where tasks are generated dynamically during the calculation. Dynamic load balancing can be used in such situations. Although it will not guarantee the decrease the SOF, it may assure that all processors will do the useful work.

References

[1] M. Baravykaitė and R. Šablinskas. The template programming of parallel algorithms. *Mathematical modelling and analysis*, **7**(1), 11 – 20, 2002.

[2] I. Foster. *Designing and Building Parallel Programs.* Addison-Wesley, 1995.

[3] A. Grama and V.Kumar A.Gupta, G.Karypis. *Introduction to Parallel Computing.* Addison Wesley, 2003.

[4] R. Horst and N.V. Thoai P.M. Pardalos. *Introduction to Global Optimization. Nonconvex optimization and its applications.* Kluwer Academic Publishers, 2000.

[5] E.W. Lawler and D.B. Smoys J.K. Lenstra, A. Rinnooy Kan. *The Traveling Salesman Problem : A Guided Tour of Combinatorial Optimization.* Wiley Series in Discrete Mathematics and Optimization. John Wiley & Sons, 1985.

[6] M. Perregaard and J. Clausen. Parallel branch and bound methods for job shop scheduling problem. *Annals of OR*, **83**, 137 – 160, 1998.

[7] A. Singh and D. Szafron J. Schaeffer. Views on template-based parallel programming. In: *CASCON '96: Proceedings of the 1996 conference of the Centre for Advanced Studies on Collaborative research.* IBM Press, 35, 1996.

[8] C. Xu and F.Lau. *Load balancing in parallel computers. Theory and Practice.* Kluwer Academic Publishers, 1997.