MATHEMATICAL MODELLING AND ANALYSIS
VOLUME 12 NUMBER 3, 2007, PAGES 277–289
 ISSN 1392-6292 print, ISSN 1648-3510 online

AN IMPLEMENTATION OF A PARALLEL GENERALIZED BRANCH AND BOUND TEMPLATE

M. BARAVYKAITĖ and R. ČIEGIS

Vilnius Gediminas Technical University

Saulėtekio al. 11, LT-10223, Vilnius, Lithuania

E-mail: {mmb,rc}@fm.vtu.lt

Received September 15, 2006; revised November 25, 2006; published online September 15, 2007

Abstract. Branch and bound (BnB) is a general algorithm to solve optimization problems. We present a template implementation of the BnB paradigm. A BnB template is implemented using C++ object oriented paradigm. MPI is used for underlying communications. A paradigm of domain decomposition (data parallelization) is used to construct a parallel algorithm. To obtain a better load balancing, the BnB template has the load balancing module that allows the redistribution of search spaces among the processors at run time. A parallel version of user's algorithm is obtained automatically.

A new derivative-free global optimization algorithm is proposed for solving nonlinear global optimization problems. It is based on the BnB algorithm and its implementation is done by using the developed BnB algorithm template library. The robustness of the new algorithm is demonstrated by solving a selection of test problems.

Key words: branch and bound, template programming, parallel algorithms

1. Problem Formulation

Many problems in engineering, physics, economics and other fields may be formulated as optimization problems, where the minimum/maximum value of an objective function should be found. Branch and bound (BnB) is a general technique to solve optimization problems. It can be used in many optimization algorithms, for example to solve combinatorial optimization or covering global optimization problems. Its general structure can be implemented as an algorithm template that will simplify the implementation of specific BnB algorithms to solve a particular problem. Similar template ideas applied for a parallel programming relieve users from doing the actual parallel programming.

Consider a minimization problem, formulated as follows

$$f^* = \min_{X \in D} f(X). \tag{1.1}$$

where $f(X)$ is an objective function, X are decision variables, and $D \subset R^n$ is a search space. Besides of the minimum f^*, one or all minimizers

$$X^* : f(X^*) = f^*$$

should be found.

The idea of the BnB algorithm is to detect the subspaces not containing the global minimizers and discard them from the further search. According to the BnB algorithm an initial approximation of the problem solution should be initiated first. The initial search space D is subsequently divided into smaller subspaces D_i. Then each subspace is evaluated trying to find out if it can contain the optimal solution. For this purpose a lower bound for the objective function $LB(D_i)$ is calculated over the subspace and compared with the upper bound $UB(D)$ for the minimum value. If $LB(D_i) \geq UB(D)$, then the subspace D_i cannot contain the global minimizer and, therefore, it is rejected from the further search. Otherwise it is inserted into the list of unexplored subspaces. The algorithm terminates when there are no subspaces in the list.

Unlike the data parallel applications (e.g. algorithms for solution of partial differential equations) optimization problems are characterized by an unpredictably varying unstructured search space [21]. This property produces additional difficulties for creation of parallel BnB algorithms: a) the change of space search order with respect to sequential one, b) processor load unbalance, c) costs of additional communications.

In this paper we describe a new implementation of a template library of parallel BnB algorithms. The results of numerical experiments are presented which show the efficiency of the presented template library. A parallel version of user's algorithm is obtained automatically from the sequential one.

A new derivative-free algorithm is proposed for solving nonlinear global optimization problems. It is based on the BnB algorithm and its implementation is done by using the BnB algorithm template library. The robustness of the new algorithm is demonstrated by solving a selection of test problems.

The rest of the paper is organized as follows. A generalized BnB algorithm is described in Section 2. In Section 3 a template based implementation of the BnB algorithm is considered. Results of computational experiments are presented in Section 4. A new black-box global optimization algorithm and its implementation by using the developed template BnB library are investigated in Section 5. Some final conclusions are done in Section 6.

2. A Generalized BnB Algorithm

The branch and bound technique is used for managing the list of sub-regions and the process of discarding and partitioning.

The general branch and bound algorithm is shown in Figure 1, where L denote a candidate set, S is the solution, $UB(D_i)$ and $LB(D_i)$ denote upper and lower bounds for minimum value of the objective function over sub-space D_i.

```
BnBAlgorithm ()
begin
  (1)  Cover solution space D by L = {L_j | D ⊆ ∪_{j=1}^{m} L_j} using covering rule
  (2)  S = ∅,  UB(D) = ∞
  (3)  while (subspace list is not empty L ≠ ∅) do
  (4)        Choose I ∈ L using selection rule, exclude I from L
  (5)        if ( LB(I) < UB(D) + ε) then
  (6)           Branch I into p subspaces I_j using branching rule
  (7)           for all (I_j, j = 1, 2, ..., p) do
  (8)                 Find UB(I_j ⋂ D) and LB(I_j) using bounding rules
  (9)                 UB(D) = min(UB(D), UB(I_j ⋂ D))
  (10)                if (LB(I_j) < UB(D) + ε) then
  (11)                   if ( I_j is a possible solution) then S = I_j
  (12)                   else L = L ∪ {I_j}
  (13)                   end if
  (14)                end if
  (15)          end for
  (16)       end if
  (17) end while
end BnBAlgorithm
```

Figure 1. General BnB algorithm.

Parallel BnB algorithms

Three main steps are performed during development of any parallel algorithm: partitioning, mapping and communication [8].

Any parallel BnB algorithm depends on distribution of the initial search space among the processors. In our BnB template a paradigm of domain decomposition (data parallelization) is used to construct a parallel algorithm. The initial search space is divided into several large subspaces that are mapped to processors and each processor performs `BBAlgorithm` independently and asynchronously. The user should decide how many subspaces are generated. The number of subspaces can coincide with or exceed the number of processors p, the decision depends on a priori knowledge of the computational complexity of subspaces. A random distribution of larger number of subspaces can improve the global load balance among processors.

A subspace is eliminated from the further search by comparing the lower bound $LB(D_i)$ for the objective function over the subspace with the upper bound $UB(D)$. The best currently found value of the objective function is used for the upper bound of the solution. In a simple version of the parallel algorithm, processors know only local values of the objective function. This can result in a slower subspace elimination. In our template processors are sharing a best known $UB(D)$. When a new value of the upper bound is found, it is broadcasted asynchronously to the other processors.

A load balancing for BnB algorithms

To obtain a better load balancing, BnB template uses the load balancing module that allows the redistribution of search spaces among the processors at run time.

The objective of the data redistribution strategies is to ensure that there exist no idle processor while others are heavily loaded, i.e to guarantee a useful work for all processors, but not to obtain the equal workload between processors [21].

The balancing module has some basic methods needed for the load balancing. A version of the diffusion load balancing algorithm is implemented as a default method [21]. The balancing process is initialized by a receiver processor. The measure of work-load is based on the number of subproblems belonging to the local list. A more accurate estimates are obtained if apriori weights are known on the complexity of the given subproblems. This information should be defined by a user of the BnB template. The full step of load balancing consists of the exchange of information among neighbours on their work-load, the selection of partners and the redistribution of subproblems among neighbour-processors. The termination of the BnB algorithm requires special protocols if the load balancing process was started. The balancing module can be extended with other balancing algorithms as well.

3. A Template Based Programming

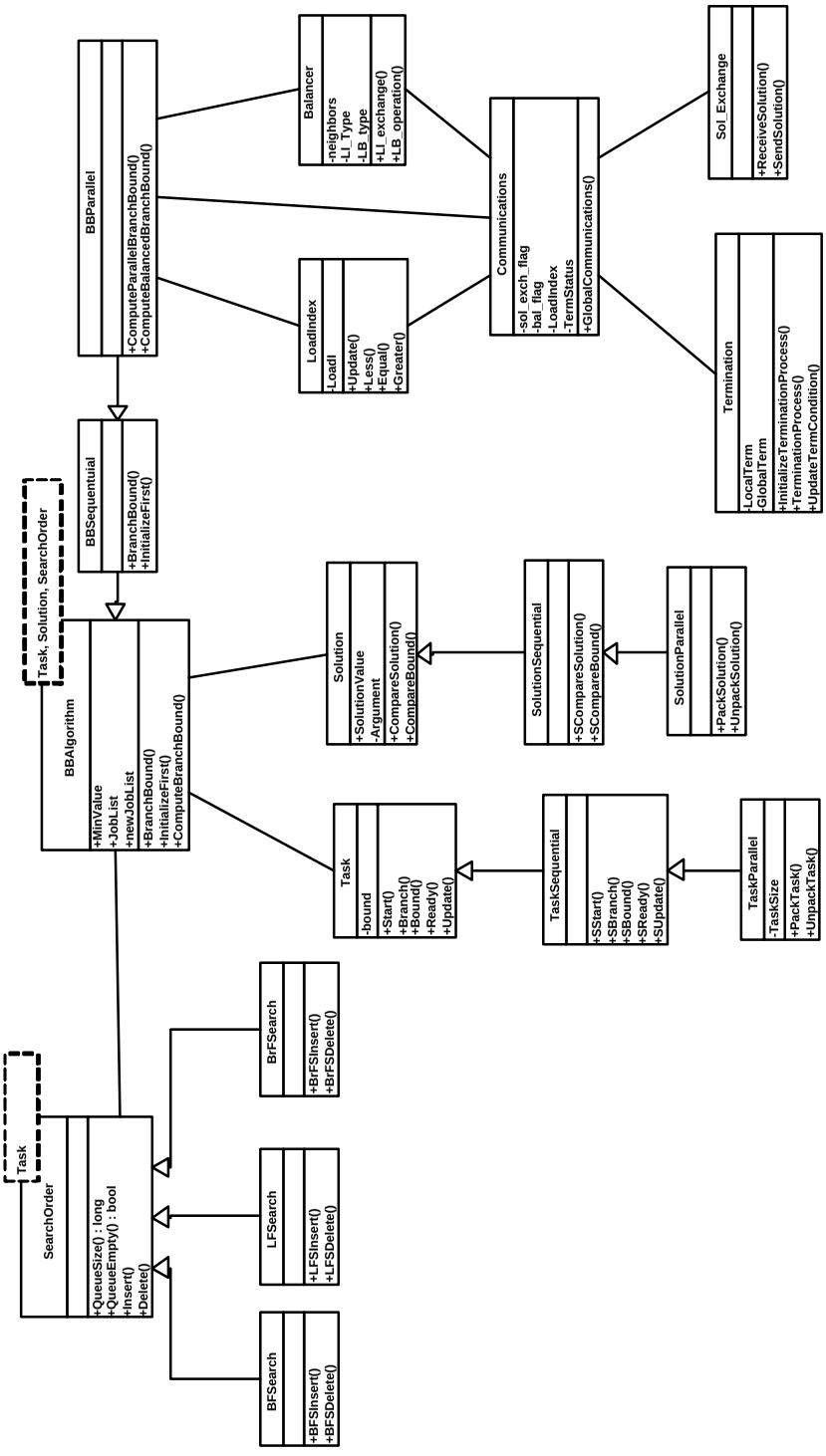
The idea of the template programming is to implement general structure of the algorithm that could be later used to solve different problems. Here we use skeleton-like templates, not data templates (or classes parameterized with data types like in STL). All general features of the algorithm and its interaction with the particular problem must be implemented in the template. The particular features related to the problem must be defined by the template user. The following steps should be implemented in any template:

- separate a problem dependent part from the general structure of the algorithm;
- implement the general structure of the algorithm as a template and use it to solve different problems.

We mention some popular examples of parallel templates used for implementation of various algorithms: Master–slaves template [16], combinatorial optimization library of software skeletons Mallba [1], CODE [18]. Templates are used very frequently for developing parallel iterative algorithms for solving systems of linear equations [3].

A template based programming is very useful in parallel programming. It was proposed by M. Cole in his PhD thesis [13], see also [6, 15, 12]. Any parallel algorithm template must fully or partially specify the main features of a parallel algorithm: partitioning, communication, agglomeration and mapping. From the user's point of view, all or nearly all coding should be sequential and almost all the parallel aspects should be provided by the tool. A parallel template is a re-usable, application-independent encapsulation of a commonly used parallel computing pattern. It is implemented as a re-usable code-template for quick and reliable development of parallel applications.

We mention examples of BnB parallelization tools BOB [5] , PICO [7], PPBB [19], PUBB [17].


BBParallel
+ComputeParallelBranchBound()
+ComputeBalancedBranchBound()
Balancer
-neighbors
-LI_Type
-LB_type
+LI_exchange()
+LB_operation()
Communications
-sol_exch_flag
-bal_flag
-LoadIndex
-TermStatus
+GlobalCommunications()
Sol_Exchange
+ReceiveSolution()
+SendSolution()
LoadIndex
-LoadI
+Update()
+Less()
+Equal()
+Greater()
Termination
-LocalTerm
-GlobalTerm
+InitializeTerminationProcess()
+TerminationProcess()
+UpdateTermCondition()
BBSequentuial
+BranchBound()
+InitializeFirst()
Task, Solution, SearchOrder
BBAlgorithm
+MinValue
+JobList
+newJobList
+BranchBound()
+InitializeFirst()
+ComputeBranchBound()
Solution
+SolutionValue
-Argument
+CompareSolution()
+CompareBound()
SolutionSequential
+SCompareSolution()
+SCompareBound()
SolutionParallel
+PackSolution()
+UnpackSolution()
Task
-bound
+Start()
+Branch()
+Bound()
+Ready()
+Update()
TaskSequential
+SStart()
+SBranch()
+SBound()
+SReady()
+SUpdate()
TaskParallel
-TaskSize
+PackTask()
+UnpackTask()
Task
SearchOrder
+QueueSize() : long
+QueueEmpty() : bool
+Insert()
+Delete()
BrFSearch
+BrFSInsert()
+BrFSDelete()
LFSearch
+LFSInsert()
+LFSDelete()
BFSearch
+BFSInsert()
+BFSDelete()

BnB algorithm template

The class scheme of the BnB algorithm template is presented in previous figure. This template implements main parts of sequential and parallel BnB algorithms. The algorithm is performed using `Task`, `Solution` and `SearchOrder` instances. The implementation of the `BBAlgorithm` is presented in the template, but users can extend this class. `SearchOrder` defines the strategies how to select the next subspace from the list of subspaces for subsequent partitioning. The most popular strategies such as the *best first* search, *last first* search and *breadth first* search are already implemented as methods and they are ready for application. The user can implement his/her own specific rules, in this case he/she should define methods `Insert, Delete, QueueSize, QueueEmpty`. Class `Task` defines the problem to be solved. It should have the basic BnB algorithm methods: `Initialize`, `Branch`, `Bound`. Some often used `Branch` methods are already implemented in the template. Standard `Bound` calculation methods (e.g. for Lipschitz functions) are included into the template. Class `Solution` implements the details of the solution and it should be implemented by the user. Class `Balancer` is used for parallel applications to balance the processor load. In order to obtain a sequential or parallel program, the user has only to select the particular `Task` and `Solution` class instances and compile the selected variant of the program. The data communication level is implemented using MPI and this level of the library is hidden from the user.

The developed template can be extended with other useful methods and algorithms, such as simulated annealing, genetic programming, the α–β search algorithm.

4. Computational Experiments

The general paradigm used to build parallel BnB algorithms is the domain decomposition, but there is a big difference between classical applications of the domain decomposition in solving PDEs and global optimization problems. Parallel optimization algorithms have an unpredictably varying unstructured search space [21]. It should be noted that because of the domain decomposition the order of search can differ for parallel and sequential branch and bound algorithms even using the same subset selection rule. Sub-spaces eliminated in the sequential algorithm can be explored in the parallel one, and it is possible that a total number of the sub-spaces searched in the parallel algorithm can be larger than in the sequential case.

Let us define the number of nodes in the generated search tree as a unit to measure the complexity of the branch and bound algorithm. We propose to estimate the growth of the number of sub-spaces in the parallel algorithm by using the following *search overhead factor*

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

where W_p is the number of sub-spaces processed in the parallel algorithm, and W_0 is the number of tasks processed by the sequential algorithm. This parameter is problem dependent, but it helps us to explain the obtained experimental results on the

efficiency of parallel algorithms, when the complexity of sub-problems is very different and the graph of generated jobs changes non-deterministically depending on the number of processors [9].

In computational experiments we minimized five Lipschitz functions with known Lipschitz constants [10]:

$$f_1(x_1, x_2) = 0.5x_1^2 - 9x_1 + 20 + 0.5x_2^2 - 9x_2 + 20,$$

$$f_2(x_1, x_2) = \frac{-1}{(x_1 - 4)^2 + (x_2 - 4)^2 + 0.7} - \frac{1}{(x_1 - 2.5)^2 + (x_2 - 3.8)^2 + 0.73},$$

$$f_3(x_1, x_2) = -\sin(2x_1 + 1) - 2\sin(3x_2 + 2),$$

$$f_4(x_1, x_2, x_3) = 100\left(x_3 - \frac{1}{4}(x_1 + x_2)^2\right)^2 + (1 - x_1)^2 + (1 - x_2)^2,$$

$$f_5(x_1, x_2, x_3) = \frac{1}{3}\sum_{i=1}^{3} x_i^2 - \prod_{i-1}^{3} \cos(10 \cdot \ln(i \cdot x_i)) + 1.$$

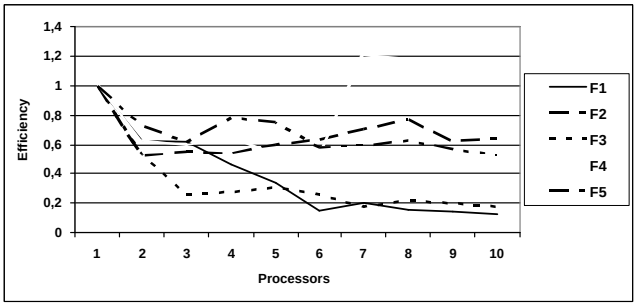
Experiments were performed on VGTU cluster *http://vilkas.vtu.lt*. It is a cluster of Pentium 4 processors which are connected by Gigabit Ethernet network (two Gigabit Smart Switch communicators). The ratio between computation and communication speeds is typical for clusters of PCs, thus the results will be even better for specialized supercomputers, such as IBM SP5.

In Figures 2 values of the efficiency coefficient and SOF of the parallel BnB algorithm are given for different numbers of processors. It follows from the presented results that the decreased efficiency can be explained by increased value of SOF coefficient. The unstructured search space varies unpredictably and it is impossible to guarantee that the efficiency of the parallel algorithm will be close to one. The load balancing helps to distribute surplus problems to free processors, but this step can also enlarge the search overhead factor.

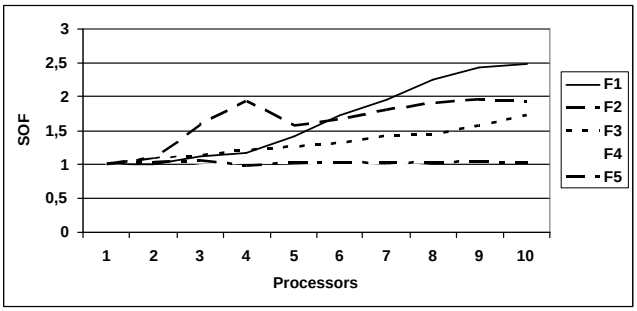
5. A Black-Box Global Optimization Algorithm

For many engineering applications only values of the objective function $f(X)$ can be computed and we do not have information on the derivatives of f (or the Lipschitz constant of this function). Thus the objective function is computed as a black-box algorithm and the gradient computation is unavailable. The target applications are simulation-based optimization problems characterized by a small number of variables (i.e., $n < 20$) and by expensive objective function evaluations (e.g. they require solution of a system of nonlinear PDEs [2]). Thus estimation of derivatives by finite differences may be prohibitively costly. A good review on derivative-free methods is given in [4].

Black-box optimization algorithms are derivative-free, only function values are required for the optimization. Parallel versions of these algorithms can greatly reduce the total solution time.



a)



b)

Figure 2. Results of computational experiments: a) the effi ciency of the parallel BnB algorithm, b) SOF of the parallel BnB algorithm.

A well-known library implementing derivative-free direct search algorithms is APPSPACK [11]. It is used for solving nonlinear unconstrained, bound-constrained, and linearly-constrained optimization problems, with possibly noisy and expensive objective functions. To find a solution of this problem, APPSPACK implements asynchronous parallel generating set search, which handles bound and linear constraints by choosing search directions that conform to the nearby boundary. Parallelism is achieved by assigning the individual function evaluations to different processors. The asynchronism enables better load balancing.

`PSwarm` tool is another example of derivative free parallel global optimization solver [20]. It is a global optimization solver for bound constrained problems (for which the derivatives of the objective function are unavailable, inaccurate or expensive). The algorithm combines pattern search and particle swarm. Basically, it applies coordinate search in the poll step and particle swarm in the search step.

We propose a new black-box global optimization algorithm, which is based on the BnB method. The algorithm is implemented by using the developed template of BnB algorithms, thus a parallel version of the algorithm is obtained automatically by running BnB code in parallel. Our algorithm is only heuristic and the main part of it depends on the definition of the bounding rule. The remaining rules are taken from

the general template of the BnB algorithm. Thus we present in detail only the outline of the bounding rule.

A bounding rule

In each sub-space D_i two sets of trial points are generated. The first set of $(2n+1)$ regular points cover the sub-space in quasi-optimal way and the remaining M points are distributed in random. We note that Sobol's sequence and the lattice rule can be used to distribute random points more uniformly.

Then a local search is done from all trial points by using the Simplex local optimization algorithm (it is a gradient-descent type method, but its implementation is derivative-free). The following three cases are considered:

1. If no local minimum points are obtained in D_i then this sub-space is eliminated from the search list L.
2. If exactly one local minimum point is obtained, the information on $UB(D)$ is updated. The sub-space is eliminated from the search list.
3. If two or more local minimum points exist in the sub-space, then a new $LB(D_i)$ estimate is computed

$$LB(D_i) = \min_i LM_i - C(\max_i LM_i - \min_i LM_i).$$

In order to increase the robustness of the algorithm up to K, local minimizers in each sub-space D_i are saved for future usage. All of them are included into the newly computed list of local minimizers. We note that this list is updated at each iteration.

Many black-box optimization algorithms suffer from serious drawback, that after rapid initial improvement of an initial approximation, the following computations give no further improvement of the solution and algorithm is stalling. This property depends mainly on the rules defining when the sub-region can be excluded from the list of promising sub-spaces. In most real–world applications it is sufficient to find fast good approximation of the global minimizer, thus we add to the algorithm two additional rules which define cases when a sub-space is eliminated from the search list L.

1. The number of sub-divisions of each initial sub-space is restricted to NS.
2. If after L subsequent steps of division the value of a best known local minimum $UB(D_i)$ is not updated, then this sub-space is excluded from the search list.

Such rules guarantee that expansive computations do not concentrate too long in some particular part of the domain D, and the whole region is tested during a reasonable time of computations.

Test functions

To assess the robustness of the new algorithm we have solved a selection of problems from [14]. The main characteristics of these problems are given in Table 1.

Table 1. The dimensions and the numbers of local and global minimizers of test functions.

Function	n	No. of local minim.	No. of global minim.
1. Rosenbbrock	2	1	1
2. McCormick	2	1	1
3. Box Betts	3	1	1
4. Paviani	10	1	1
5. Generalized Rosenbrock	15	1	1
6. Gold and Price	2	4	1
7. Shekel5	4	5	1
8. Shekel7	4	7	1
9. Shekel10	4	10	1
10. Levy4	4	71000	1
11. Levy5	5	10^5	1
12. Levy5	6	10^6	1
13. Levy7	7	10^8	1
14. Griewank	10	1000	1
15. Six Hump Camel	2	6	2
16. Branin	2	23	5
17. Shubert	2	400	9
18. Hansen	2	760	9

Results

Up to 32 processors were used to solve each problem. For any number of processors the BnB algorithm converged to the optimal solution for problems 1-13 and 15-18. In the case of Griewank's problem the accuracy of the computed solution depended on the specified domain D. If we take $D = [-5, 7] \times \cdots \times [-5, 7]$, then the optimal global minimizer is obtained. For $D = [-50, 70] \times \cdots \times [-50, 70]$, we have computed only an approximation of the exact minimizer. Since this problem is multidimensional and it has a very large number of local minimizers, the computed $LB(D_i)$ bounds were not very accurate and subproblems were eliminated from the search list L mainly according to the two additional rules, given above. We note that an improvement of the accuracy was obtained by increasing the number of possible divisions and number M of at random distributed initial approximations.

The comparison of the new `BlackBox` algorithm with `Appspack` and `PSwarm` algorithms is presented. First sequential versions of these algorithms were compared. Results of calculations are presented in Table 2. In many cases `BlackBox` performed better than `PSwarm` and in cases with several global minima it also outperformed the `Appspack` algorithm.

The parallel version of `BlackBox` algorithm was obtained using BnB algorithm template. The speed-up for the Levy, Griewank and Generalized Rosenbrock problems was measured and is presented in Table 3. In case of Levy4 function a good speed-up is achieved. In other cases the speed-up is worse than speed-ups obtained

Table 2. The comparison of sequential execution time.

Function	Appspack	PSwarm	Black Box
Rosenbbrock	0.03	0.01	0.01
McCormick	2.02	1.52	0.02
Box Betts	0.05	0.05	0.06
Paviani	11.02	2.5	10.96
Generalized Rosenbrock	35.6	170.2	129.19
Gold and Price	0.23	3.5	0.02
Shekel5	0.33	3.6	0.45
Shekel7	3.6	5.0	0.59
Shekel10	5.7	8.1	0.65
Levy4	13.6	25.9	18.13
Levy5	23.35	42.5	29.24
Levy5	32.4	83.6	74.32
Levy7	49.6	535.8	357.63
Griewank	40.9	92.1	77.46
Six Hump Camel	2.1	0.5	0.06
Branin	2.0	0.2	0.1
Shubert	2.3	3.54	1.27
Hansen	2.9	0.95	0.8

using Appspack (see Table 4) and better that ones reached using PSwarm (see Table 5) for the same problems.

Table 3. The speedup of the parallel Black Box algorithm.

Processors	GenRos	Levy4	Levy5	Levy6	Levy7	Griewank
1	1	1	1	1	1	1
2	1.81	2.05	2.01	2.78	1.62	1.78
3	0.37	2.96	2.42	1.92	2.51	1.49
4	0.34	5.51	2.89	2.66	2.11	1.28
8	0.51	6.73	3.96	4.02	2.30	0.98
16	0.69	14.89	5.49	4.54	3.67	1.22
32	1.29	30.82	5.73	7.63	5.94	1.39

6. Conclusions

In this study we have presented a description of a new template of parallel BnB algorithms. It presents `C++` classes for all main steps of BnB algorithm including

Table 4. The speedup of the parallel Appspack algorithm.

Processors	GenRos	Levy4	Levy5	Levy6	Levy7	Griewank
1	1	1	1	1	1	1
2	1.17	1.34	1.85	1.84	1.57	1.43
3	2.117	1.40	2.53	2.12	2.17	2.33
4	2.89	2.0	3.69	2.74	2.43	4.59
8	5.74	3.24	5.86	4.90	3.01	7.57
16	10.78	4.38	9.32	7.71	4.59	10.76
32	16.95	9.06	31.07	12.96	11.81	22.72

Table 5. The speedup of the parallel PSwarm algorithm.

Processors	GenRos	Levy4	Levy5	Levy6	Levy7	Griewank
1	1	1	1	1	1	1
2	0.90	0.97	1.41	1.66	1.30	1.82
3	1.17	0.88	0.94	1.30	1.17	2.05
4	1.30	1.07	0.90	1.25	0.95	1.91
8	1.69	1.13	0.94	1.54	1.122	2.83
16	2.11	1.27	0.95	1.94	1.26	3.01
32	3.39	1.04	1.05	1.24	1.23	3.28

many examples for the selection, bound estimation and branching steps. A parallel version of user's algorithm is obtained automatically. The load balancing level of the BnB algorithm template implements a variant of the diffusion method. The numerical experiments have shown the efficiency of the template library.

A new derivative-free algorithm is proposed for solving nonlinear global optimization problems. It is based on the BnB method and its implementation is done by using the developed BnB algorithm template. The robustness of the new algorithm is demonstrated by solving a selection of test problems. No analysis is still done to tune parameters C, M, NS of the proposed algorithm, this question is still open.

Acknowledgment

This work was supported by the Lithuanian State Science and Studies Foundation within the project on B-03/2007 "Global optimization of complex systems using high performance computing and GRID technologies" and by the Eureka Project EUREKA E!3691 OPTCABLES.

References

[1] E. Alba, F. Almeida and at all. *Mallba: A library of skeletons for combinatorical optimization*. Technical report, 2001.

[2] M. Baravykaitė, R. Belevičius and R. Čiegis. One application of the parallelization tool of master – slave algorithms. **13**(4), 393–404, 2002.

[3] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, Ch. Romine and Henk van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, Philadelphia PA, 1994.

[4] A.R. Conn, K. Scheinberg and Ph.L. Toint. Recent progress in unconstrained nonlinear optimization without derivatives. *Mathematical programming*, **79**, 397–414, 1997.

[5] B. Le Cun and C. Roucairol. *Bob: a unifi ed platform for implementing branch-and-bound like algorithms. Technical Report 95/16 sep.* Université de Versailles - Laboratoire PR*i*SM, 1995.

[6] I. Dorta, C. Leon and C. Rodriquez. Parallel branch and bound skeletons: message passing and shared memory implementations. In: *Proceedings of PPAM 2003, LNCS*, volume 3019. Springer, 286–291, 2003.

[7] J. Eckstein, W.E. Hart and C.A. Phillips. *Pico: An object-oriented framework for parallel branch and bound, rutcor research report. Technical Report 40-2000*. Rutgers University, Piscataway, NJ, 2000.

[8] I. Foster. *Designing and building parallel programs*. Addison-Wesley, 1995.

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

[10] P. Hansen and B. Jaumard. Lipschitz optimization. In: *Handbook of Global Optimization, volume 2 of Nonconvex Optimization and Its Applications*. Kluwer Academic Publishers, Dodrecht, 404–493, 1995.

[11] T.G. Kolda. Revisiting asynchronous parallel pattern search for nonlinear optimization. *SIAM Journal on Optimization*, **16**(2), 563–586, 2005.

[12] H. Kuchen. A skeleton library. In: *Proceedings of Euro-Par 2002, LNCS vol.2400*. Springer-Verlag, 620–629, 2002.

[13] title= M. I. Cole.

[14] K. Madsen and J. Žilinskas. *Testing of attraction based subdivision and interval methods for global optimization.* IMM-REP-*2000-04*. Department of Mathematical Modelling, Technical University of Denmark, Lyngby, Denmark, 2000.

[15] B. Preiss, D. Goswami and A. Singh. From design patterns to parallel architecture skeletons. *Journal of Parallel and Distributed Computing*, **62**(4), 669–695, 2002.

[16] R. Šablinskas. *Investigation of algorithms for distributed memory parallel computers*. 1999. PhD thesis

[17] Y. Shianno and T. Fujier. *Pubb (parallelization utility for branch-and-bound algorithms). User manual*. Technical Report, Version 1.0, 1999.

[18] A. Singh, D. Szafron and J. Schaeffer. Views on template-based parallel programming. In: *CASCON 96 CDRom Proceedings*, Toronto, October, 1996.

[19] S. Tschoke and T. Polzer. *Portable parallel branch-and-bound library ppbb-lib. user manual. Technical Report Version 2.0.*

[20] A. I. F. Vaz and L. N. Vicente. *A Particle Swarm Pattern Search Method for Bound Constrained Nonlinear Optimization. Technical Report 06-08.* Department of Mathematics, University of Coimbra, Portugal, 2006.

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