ParaLEX: A Parallel Extension for the CPLEX Mixed Integer Optimizer*

Yuji Shinano[1] and Tetsuya Fujie[2]

[1] Division of Systems and Information Technology, Institute of Symbiotic Science and Technology, Tokyo University of Agriculture and Technology, 2-24-16, Naka-cho, Koganei-shi, Tokyo 184-8588, Japan
yshinano@cc.tuat.ac.jp
[2] School of Business Administration, University of Hyogo, 8-2-1, Gakuen-nishimachi, Nishi-ku, Kobe 651-2197, Japan
fujie@biz.u-hyogo.ac.jp

Abstract. The ILOG CPLEX Mixed Integer Optimizer is a state-of-the-art solver for mixed integer programming. In this paper, we introduce ParaLEX which realizes a master-worker parallelization specialized for the solver on a PC cluster using MPI. To fully utilize the power of the solver, the implementation exploits almost all functionality available in it. Computational experiments are performed for MIPLIB instances on a PC cluster composed of fifteen 3.4GHz pentiumD 950 (with 2G bytes RAM) PCs (running a maximum of 30 CPLEX Mixed Integer Optimizers). The results show that ParaLEX is highly effective in accelerating the solver for hard problem instances.

Keywords: Mixed Integer Programming, Master-Worker, Parallel Branch-and-cut.

1 Introduction

The ILOG CPLEX Mixed Integer Optimizer [6] is one of the most successful commercial codes for MIP (Mixed Integer Programming). MIP problem is to optimize (minimize or maximize) a linear function subject to linear inequalities and/or linear equalities with the restriction that some or all of the variables must take integer values. MIP has a wide variety of industrial, business, science and educational applications. In fact, recent remarkable progress of MIP optimizers, including CPLEX, leads to the increased importance of MIP models. CPLEX has continued to incorporate computational improvements into a branch-and-cut implementation, which results an efficient and robust code. It involves both standard and advanced techniques such as preprocessing, many kinds of cutting planes, heuristics, and strong branching. Due to limited space, we omitted MIP-related from this paper (see [8,11] for MIP models and algorithms). In [2], recent software systems for MIP are introduced. Many researchers have developed

* This work was partially supported by MEXT in Japan through Grants-in-Aid(18510118).

parallelization frameworks of branch-and-bound and branch-and-cut algorithms, including ALPS/BiCeOS, PICO, SYMPHONY, BCP, PUBB. See recent surveys [3,4,9].

In this paper, we propose a master-worker parallelization of CPLEX, named ParaLEX (Parallel extension for CPLEX MIP optimizer). The main feature of our implementation is that it fully utilizes the power of CPLEX : The entire search tree (or, branch-and-cut tree) produced by ParaLEX is composed of subtrees produced by CPLEX on master and workers. ParaLEX has a simpler structure than our previous work of a parallelization of CPLEX using the PUBB2 framework [10].

2 ParaLEX

In this section, we introduce ParaLEX briefly. We had three major goals for the design of ParaLEX.

– Most of all the functionality of CPLEX must be available.
– Future versions of CPLEX must be "parallelizable" without any code modification needed.
– Parallel implementation must be as simple as possible.

To achieve these design goals, ParaLEX is

– composed of two types of solvers, each of which runs CPLEX with almost full functionality,
– is implemented in C++ but the most primitive CPLEX Callable Library is used, and
– is essentially a simple Master-Worker parallelization.

The branch-and-cut algorithm is an enumerative algorithm which repeatedly partitions a given problem instance into several subproblems. The algorithm therefore forms a tree called search tree or branch-and-cut tree. Subproblems will be also referred to as nodes in the subsequent of this paper. Similarly, a given problem instance will be also referred to as a root node. Next, we introduce the notion of the *ParaLEX instance* which comprises a preprocessed problem instance and global cuts applied to the root node. Preprocessing and generation of global cuts (or, cutting planes), both of which are applied to the root node, are quite effective in CPLEX in giving a tighter reformulation of MIP even though they are time consuming. It is certain that, if we drop these functionalities in parallelization, the solution time of MIP becomes greater than that by a sequential CPLEX solver. On the other hand, if these functionalities are performed from scratch on each PE (Processing Element), the parallelization cannot achieve high performance in general. ParaLEX instances are introduced based on the observation. A subproblem representation used in ParaLEX is the difference between a node LP object in the branch callback routine of CPLEX and the ParaLEX instance.

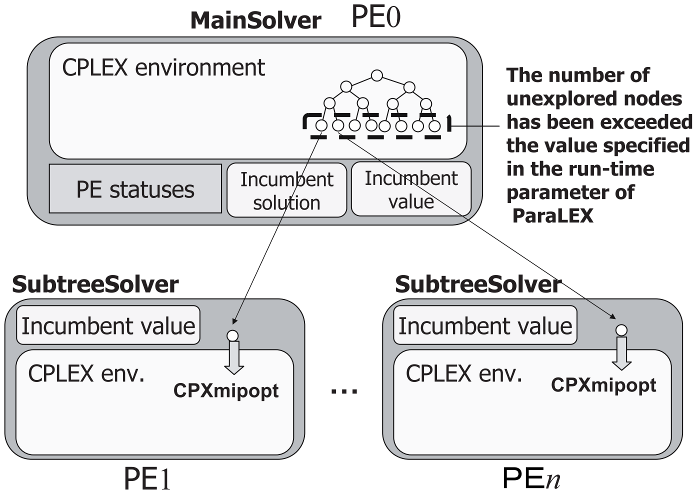

Fig. 1. Initialization phase of ParaLEX

ParaLEX is implemented as a SPMD-type MPI program[1] but is composed of the following two types of objects (solvers) depending on the rank of the process, which we call the *PE rank*.

MainSolver (PE rank $= 0$**).** This solver reads an original problem instance data, and then performs preprocessing, and generates global cuts to create the ParaLEX instance. It not only manages the PEs on the system but also solves nodes by applying the `CPXmipopt` function of CPLEX. It keeps an incumbent solution which is the best solution found so far among all the PEs and, at the end of computation, outputs an optimal solution.

SubtreeSolver (PE rank ≥ 1**).** This solver first receives the ParaLEX instance from the MainSolver. It also receives nodes from the MainSolver or other SubtreeSolvers. The nodes received are solved by CPLEX. If an improved incumbent solution is found in this solver, the solution is sent to the MainSolver.

At first, the MainSolver starts solving the ParaLEX instance by CPLEX. When the number of unexplored nodes in the CPLEX environment has been exceeded the threshold value given by the ParaLEX run-time parameter, the MainSolver starts distributing its unexplored nodes one by one to SubtreeSolvers. In our computational experiments reported in Section 3, we set the threshold parameter as $\min\{200, 5.0/(\text{average time to compute one node in the sequential run})\}$. Figure 1 shows the initialization phase.

When ParaLEX runs with several PEs, a search tree is partitioned into several subtrees produced by the corresponding PEs. Each subtree is maintained in the

[1] The Master-Worker program can also be implemented as an MPMD-type MPI program. However, we have been familiar with an SPMD implementation due to the requirement of MPI-1 functionality.

CPLEX environment of the corresponding PE (MainSolver or SubtreeSolver). In normal running situations, communications between the MainSolver and the SubtreeSolver and between the SubtreeSolvers are done in the branch callback routine of CPLEX. In this callback routine, the PE from which the callback is called sends a node to a PE if necessary. The node is then solved as a problem instance by CPLEX. Note that the node transferred is solved twice, once in the sender as a node and once in the receiver as a root node. In the receiver side, however, the root node is not solved from scratch, because its LP (Linear Programming) basis generated in the sender side is also transferred and is used as the starting basis. Though it is solved as a root node, it may be pruned without any branch by trying to apply extra cuts to the node. Eventually, it may lead to reduced total computation times.

The MainSolver maintains statuses of all the PEs. Each SubtreeSolver notifies to the MainSolver the number of unexplored nodes in the CPLEX environment and the best bound value among these nodes as the PE status, when the number of nodes processed from the previous notification has been exceeded the threshold value given by the ParaLEX run-time parameter. When a PE has finished solving an assigned node, it becomes an idle solver. If the MainSolver detects an idle solver, it sends a subproblem-transfer request message to a PE which has an unexplored node with the best bound value by referring to the PE statuses. There is a delay updating the PE statuses in the MainSolver. Therefore, the PE which receives the subproblem-transfer request may not have enough many nodes in its CPLEX environment. In such a case, the solver rejects the request. If the solver has more nodes than the threshold value given by the ParaLEX run-time parameter, it accepts the request and sends a node to the destination PE that is indicated in the request message. Note that at most one node can be transferred in one request message. Figure 2 shows a message sequence for the subproblem transfer.

The node to be transferred can be selected in the selection callback routine of CPLEX. When the node with the best bound transfer is specified in the ParaLEX run-time parameter, the best bound node is selected in this callback routine. The selection is done by a computationally intensive linear search of the unexplored nodes in the CPLEX environment. On the other hand, when a default selection is specified in this parameter, a node is selected according to the selection rule specified in the CPLEX run-time parameter. In this case, the unexplored nodes are arranged by the selection rule order in the CPLEX environment, and thus selection does not take a long time.

When an improved solution is found in the MainSolver, its incumbent value is sent to all the SubtreeSolvers. When an improved solution is found in the SubtreeSolver, the incumbent solution is sent to the MainSolver and the MainSolver sends the incumbent value to all the SubtreeSolvers. Improved incumbent solutions are detected in the incumbent callback routine of CPLEX and its notification is done in the branch callback of CPLEX.

The `CPXmipopt` function that solves a problem instance is suspended after the number of nodes specified in the ParaLEX run-time parameter has

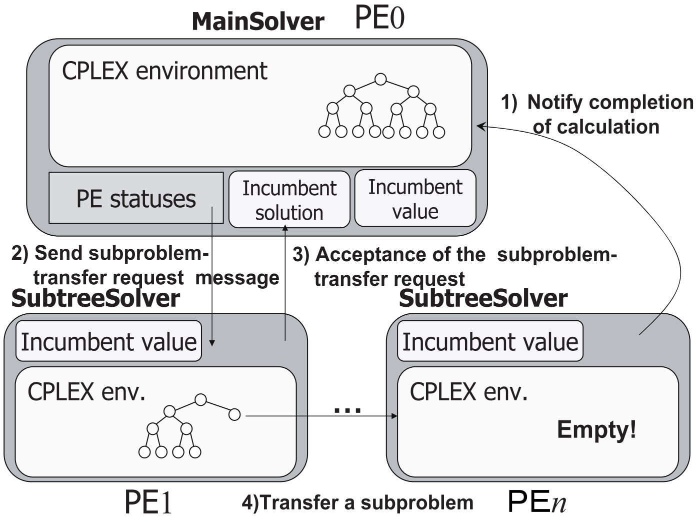

Fig. 2. Message sequence for the subproblem transfer

been processed. When suspended time comes, the latest incumbent value is set on CPLEX environment using the `CPXsetdblparam` function of CPLEX with `CPX_PARAM_CUTUP` (if the problem is minimization) or with `CPX_PARAM_CUTLO` (if the problem is maximization). After that, the suspension is resumed. Therefore, there is a delay in the notification of the incumbent value.

3 Computational Experiments

In this section, we report our computational results. We used a PC cluster composed of fifteen 3.4GHz pentiumD 950 (with 2Gbytes RAM) PCs connected with Gigabit Ethernet, where 30 CPLEX Mixed Integer Optimizers (version 10.1) were available. The MPI library used is mpich2-1.0.5p4. Problem instances were selected from the MIPLIB2003 library[2][1] which is a standard test set to evaluate the performance of MIP optimizers.

We first discuss the effects of the ParaLEX run-time parameters described in Section 2.

– MIP emphasis indicator `CPX_PARAM_MIPEMPHASIS` (CPLEX) : This is a parameter prepared by CPLEX to tell the solver whether it should find a feasible solution with high quality or prove the optimality. Among several options, we selected the following ones, which are concerned with the optimality.
  - `CPX_MIPEMPHASIS_BALANCED` : Balance optimality and integer feasibility (default parameter of CPLEX).
  - `CPX_MIPEMPHASIS_OPTIMALITY` : Emphasizing optimality over feasibility.

[2] URL: http://miplib.zib.de

  - `CPX_MIPEMPHASIS_BESTBOUND` : Emphasizing moving best bound.
- Subproblem transfer mode (ParaLEX) : This is a parameter for PEs to transform a node.
  - `best_bound_transfer` : The best bound node is selected.
  - `cplex_default_transfer` : A node is selected according to the CPLEX run-time parameter.
- `CPXmipopt` interval (ParaLEX) : This is a parameter for PEs to specify the suspended time of `CPXmipopt`.
  - `fixed_iter` : Every time `CPXmipopt` terminates with the node limit of this value to check a subproblem-request from an another PE. In this paper, we set `fixed_iter` = 20.
  - `fixed_time` : By a sequential run, we estimate the number of nodes generated by `CPXmipopt` during the time `fixed_time`. Then, this estimated value is used for the node limit of `CPXmipopt`. In this paper, we set `fixed_time` = 1 (sec.).

Table 1 displays the parallel speedups obtained over 5 runs for the easy instances `fast0507` and `mas74` with possible combinations of the following two parameters, the Subproblem transfer mode and the `CPXmipopt` interval. `CPX_PARAM_MIPEMPHASIS` is set to `CPX_MIPEMPHASIS_BALANCED`. As the table shows, it is hard to determine the most suitable combination of the parameter values. On the other hand, `cplex_default_transfer` is competitive or better than `best_bound_transfer`, which indicates that a PE which receives a transferred node should continue the branch-and-cut search along with the parent PE (i.e., the PE which sends the node). Hence, we selected `cplex_default_transfer` for the subsequent computations. We observed that the effect of the `CPXmipopt` interval depends on the behavior of a sequential run of CPLEX since computing time per node varies considerably with problem instances. Hence, we decided to use `fixed_time`.

Table 2 shows the results for the `CPX_PARAM_MIPEMPHASIS` parameter. The results are obtained over 5 runs except that the `noswot` instance is examined over 1 run. We note that the computing time of sequential run varies with this parameter, and the most suitable parameter-choice also varies with the problem instance. From the table, we see that superlinear speedup results are obtained for several problem instances. In particular, noteworthy speedup is obtained for the `noswot` instance. We observed that superlinear speedup occurs when a good feasible solution is hardly obtained by CPLEX. In this case, a parallel search could find a good feasible solution faster than a sequential search. Since fast finding of a good feasible solution can lead to pruning of many unexplored nodes, the parallel search could generate a smaller search tree than the sequential search. Therefore, computation is performed to a smaller number of nodes by many PEs and, as a result, the superlinear speedup is obtained. We also observe that high speedup results are obtained for `CPX_MIPEMPHASIS_OPTIMALITY`.

Finally, we report the results for hard problem instances with 30 PEs. We continue to use the parameters `cplex_default_transfer` and `fixed_time`. `CPX_PARAM_MIPEMPHASIS` was determined by an observation of the behavior of upper and lower bounds within several hours from the beginning and by the information in the MIPLIB 2003 website.

Table 1. Parallel speedups for easy instances (CPX_MIPEMPHASIS_BALANCED)

	# of PEs					
	2	3	5	10	20	30
fast0507, best_bound_transfer, fixed_iter : 880.61 sec. (sequential)						
ave.	0.45	1.16	1.38	1.98	2.96	3.07
min.	0.45	1.13	1.37	1.97	2.93	3.04
max.	0.45	1.21	1.39	1.99	3.00	3.13
fast0507, best_bound_transfer, fixed_time : 880.61 sec. (sequential)						
ave.	0.99	1.04	1.33	3.07	3.56	3.71
min.	0.99	1.03	1.30	3.03	3.51	3.68
max.	0.99	1.05	1.42	3.21	3.63	3.78
fast0507, cplex_default_transfer, fixed_iter : 880.61 sec. (sequential)						
ave.	0.87	1.11	0.61	2.36	3.39	3.15
min.	0.76	1.10	0.61	2.30	3.37	3.13
max.	0.96	1.12	0.61	2.52	3.43	3.15
fast0507, cplex_default_transfer, fixed_time : 880.61 sec. (sequential)						
ave.	1.23	1.15	1.36	1.56	2.41	2.71
min.	1.23	1.04	1.31	1.51	2.11	2.65
max.	1.24	1.36	1.44	1.72	2.92	2.75
mas74, best_bound_transfer, fixed_iter : 1292.99 sec. (sequential)						
ave.	0.04	0.06	0.28	1.39	1.84	0.50
min.	0.03	0.05	0.21	1.29	1.19	0.34
max.	0.06	0.06	0.33	1.64	4.36	0.64
mas74, best_bound_transfer, fixed_time : 1292.99 sec. (sequential)						
ave.	0.12	0.04	0.25	1.40	3.21	0.35
min.	0.11	0.04	0.24	1.32	1.59	0.30
max.	0.12	0.04	0.27	1.45	5.40	0.39
mas74, cplex_default_transfer, fixed_iter : 1292.99 sec. (sequential)						
ave.	0.57	0.51	1.09	1.38	5.21	8.36
min.	0.46	0.34	0.84	1.11	4.43	7.14
max.	0.67	0.86	1.38	2.26	6.32	10.77
mas74, cplex_default_transfer, fixed_time : 1292.99 sec. (sequential)						
ave.	0.82	0.91	1.34	1.90	4.66	10.01
min.	0.80	0.78	1.22	1.79	4.39	8.62
max.	0.85	1.09	1.49	2.08	5.11	13.98

– a1c1s1 : 142960.36 (sec.) with CPX_MIPEMPHASIS_BESTBOUND
– arki001 : 3924.87 (sec.) with CPX_MIPEMPHASIS_OPTIMALITY
– glass4 : 2001.50 (sec.) with CPX_MIPEMPHASIS_BALANCED
– roll3000 : 173.73 (sec.) with CPX_MIPEMPHASIS_BESTBOUND
– atlanta-ip : 2512451.70 (sec.) with CPX_MIPEMPHASIS_BALANCED

These problem instances have been solved to optimality recently [5,7], and they are still hard to be solved with sequential and default solver strategies. Actually, the roll3000 instances cannot be solved sequentially within 230464.65 sec. with CPX_MIPEMPHASIS_BESTBOUND. 2213435 nodes remain unexplored. Hence,

Table 2. Parallel speedups for easy instances (`cplex_default_transfer`, `fixed_time`)

	# of PEs					
	2	3	5	10	20	30
`fast0507`, `CPX_MIPEMPHASIS_BALANCED` : 880.61 sec. (sequential)						
ave.	1.23	1.15	1.36	1.56	2.41	2.71
min.	1.23	1.04	1.31	1.51	2.11	2.65
max.	1.24	1.36	1.44	1.72	2.92	2.75
`fast0507`, `CPX_MIPEMPHASIS_OPTIMALITY` : 10704.98 sec. (sequential)						
ave.	9.70	14.19	14.93	15.53	15.52	15.53
min.	9.63	13.73	14.71	15.50	15.49	15.50
max.	9.77	14.65	15.15	15.56	15.53	15.56
`fast0507`, `CPX_MIPEMPHASIS_BESTBOUND` : 15860.45 sec. (sequential)						
ave.	0.47	0.63	0.72	0.71	2.03	2.48
min.	0.44	0.48	0.72	0.68	2.01	2.45
max.	0.65	0.69	0.73	0.74	2.07	2.52
`mas74`, `CPX_MIPEMPHASIS_BALANCED` : 1292.99 sec. (sequential)						
ave.	0.82	0.91	1.34	1.90	4.66	10.01
min.	0.80	0.78	1.22	1.79	4.39	8.62
max.	0.85	1.09	1.49	2.08	5.11	13.98
`mas74`, `CPX_MIPEMPHASIS_OPTIMALITY` : 1759.18 sec. (sequential)						
ave.	1.48	1.52	1.47	2.14	7.48	11.23
min.	1.40	1.46	1.41	1.91	6.86	10.07
max.	1.56	1.64	1.62	2.27	7.86	13.09
`mas74`, `CPX_MIPEMPHASIS_BESTBOUND` : 3674.92 sec. (sequential)						
ave.	1.02	0.96	0.94	1.02	1.01	1.04
min.	1.01	0.95	0.81	0.97	0.97	0.91
max.	1.03	0.98	1.04	1.06	1.05	1.21
`harp2`, `CPX_MIPEMPHASIS_BALANCED` : 5586.38 sec. (sequential)						
ave.	8.93	5.15	8.28	8.98	13.96	24.94
min.	5.34	3.24	5.90	3.15	8.58	17.67
max.	13.70	22.00	13.55	35.23	24.76	40.86
`harp2`, `CPX_MIPEMPHASIS_OPTIMALITY` : 2014.80 sec. (sequential)						
ave.	2.16	1.87	2.53	2.47	4.15	24.16
min.	1.53	1.10	2.05	1.61	2.49	20.61
max.	3.34	2.71	4.34	6.16	7.34	32.02
`harp2`, `CPX_MIPEMPHASIS_BESTBOUND` : 2974.34 sec. (sequential)						
ave.	1.43	2.62	3.10	2.71	4.70	8.22
min.	1.24	1.88	2.12	1.78	3.11	4.03
max.	1.59	3.41	6.95	5.67	8.17	18.64
`noswot`, `CPX_MIPEMPHASIS_OPTIMALITY` : 38075.95 sec. (sequential)						
	–	–	–	27.01	119.61	276.03
`noswot`, `CPX_MIPEMPHASIS_BESTBOUND` : 193373.54 sec. (sequential)						
	–	–	–	10.58	17.44	71.02

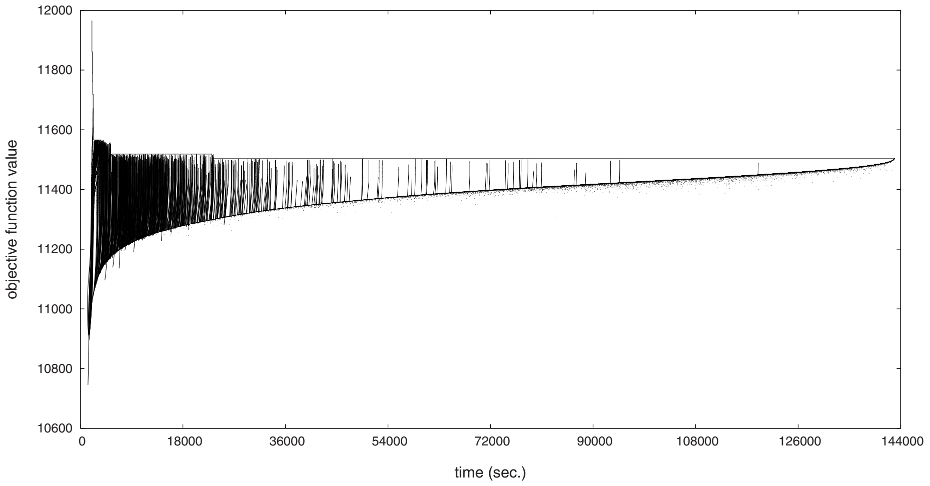

Fig. 3. Upper and lower bounds for the `a1c1s1` instance

we have a superlinear speedup result for this problem instance. We remark that R. Miyashiro reports that the `roll3000` instance were solved in about half a day with `CPX_MIPEMPHASIS_BESTBOUND` (see the MIPLIB2003 website). However, the computing environment is different: He used a 32-bit PC while we used a 64-bit machine, and the mipgap tolerances he used are different from ours (private communication).

In Figure 3, we draw upper and lower bounds as functions of time for the `a1c1s1` instance.

4 Concluding Remarks

The development of ParaLEX is still in its preliminary stages. In the initial design of ParaLEX, we intended to transfer nodes to the SubtreeSolvers only from the MainSolver or via the MainSolver to simplify the communication mechanism. However, this did not work well in many cases. Hence, we modified the transfer sequence as described in this paper. It is still a simple concept, but its implementation becomes complicated. We aim to reconsider the mechanism using what was learned in this development.

ParaLEX was originally developed by using CPLEX 9.0, but it could be compiled with CPLEX 10.1 without any modification of the codes related to the CPLEX Callable Library. Using CPLEX 9.0, ParaLEX solved the `rout` instance about 66 times faster than the `cplex` command of the version did. However, CPLEX 10.1 can solve the instance about 22 times faster than CPLEX 9.0. We observed that ParaLEX with CPLEX 10.1 is not so attractive for the `rout` instance. To solve MIP problems faster, algorithmic improvements may be more significant than that by using parallelization. On the other hand, it is still quite hard to find a good feasible solution for some problem instances. If fast finding of a good feasible solution becomes possible by using new heuristic algorithms or different search strategies, this could lead to a tremendous speedups.

However, ParaLEX is still very attractive to solve much harder instances for CPLEX 10.1. The significance of our approach is that ParaLEX has a potential to accelerate the latest version of CPLEX using parallelization. The parallel search itself is a method to obtain a good feasible solution faster than the sequential search. This is a reason why ParaLEX achieved superlinear speedups for several problem instances. Moreover, by only using the latest version with parallelization (note that we did not use any special structure of problem instances), several of today's hardest instances of MIPLIB were solved to optimality. Therefore, ParaLEX is highly effective to solve hard problem instances.

References

1. Achterberg, T., Koch, T., Martin, A.: MIPLIB 2003. Oper. Res. Lett. 34, 361–372 (2006)
2. Atamtürk, A., Martin, W.P., Savelsbergh, M.W.P.: Integer-Programming Software Systems. Ann. Oper. Res. 140, 67–124 (2005)
3. Bader, D.A., Hart, W.E., Phillips, C.A.: Parallel Algorithm Design for Branch and Bound. In: Greenberg, H.J. (ed.) Tutorials on Emerging Methodologies and Applications in Operations Research, ch. 5, Kluwer Academic Press, Dordrecht (2004)
4. Crainic, T., Le Cun, B., Roucairol, C.: Parallel Branch-and-Bound Algorithms. In: Talbi, E. (ed.) Parallel Combinatorial Optimization, ch. 1, Wiley, Chichester (2006)
5. Ferris, M.: GAMS: Condor and the grid: Solving hard optimization problems in parallel. Industrial and Systems Engineering, Lehigh University (2006)
6. ILOG CPLEX 10.1 User's Manual, ILOG, Inc. (2006)
7. Laundy, R., Perregaard, M., Tavares, G., Tipi, H., Vazacopoulos, A.: Solving Hard Mixed Integer Programming Problems with Xpress-MP: A MIPLIB 2003 Case Study. Rutcor Research Report 2-2007, Rutgers University (2007)
8. Nemhauser, G.L., Wolsey, L.A.: Integer Programming and Combinatorial Optimization. John Wiley & Sons, New York (1988)
9. Ralphs, T.K.: Parallel Branch and Cut. In: Talbi, E. (ed.) Parallel Combinatorial Optimization, ch. 3, Wiley, Chichester (2006)
10. Shinano, Y., Fujie, T., Kounoike, Y.: Effectiveness of Parallelizing the ILOG-CPLEX Mixed Integer Optimizer in the PUBB2 Framework. In: Kosch, H., Böszörményi, L., Hellwagner, H. (eds.) Euro-Par 2003. LNCS, vol. 2790, pp. 451–460. Springer, Heidelberg (2003)
11. Wolsey, L.A.: Integer Programming. John Wiley & Sons, New York (1998)