Konrad-Zuse-Zentrum
für Informationstechnik Berlin

Takustraße 7
D-14195 Berlin-Dahlem
Germany

YUJI SHINANO
TOBIAS ACHTERBERG*
TIMO BERTHOLD**
STEFAN HEINZ**
THORSTEN KOCH

ParaSCIP – a parallel extension of SCIP

* ILOG, on IBM Deutschland GmbH, Ober-Eschbacher Str. 109, 61352 Bad Homburg v.d.H., Germany
** Supported by the DFG Research Center MATHEON *Mathematics for key technologies* in Berlin.

ZIB-Report 10-27 (December 2010)

ParaSCIP – a parallel extension of SCIP*

Yuji Shinano, Tobias Achterberg, Timo Berthold, Stefan Heinz, and Thorsten Koch

Abstract *Mixed integer programming (MIP)* has become one of the most important techniques in Operations Research and Discrete Optimization. SCIP (Solving Constraint Integer Programs) is currently one of the fastest non-commercial MIP solvers. It is based on the *branch-and-bound* procedure in which the problem is recursively split into smaller subproblems, thereby creating a so-called *branching tree*. We present ParaSCIP, an extension of SCIP, which realizes a parallelization on a distributed memory computing environment. ParaSCIP uses SCIP solvers as independently running processes to solve subproblems (nodes of the branching tree) locally. This makes the parallelization development independent of the SCIP development. Thus, ParaSCIP directly profits from any algorithmic progress in future versions of SCIP. Using a first implementation of ParaSCIP, we were able to solve two previously unsolved instances from MIPLIB2003, a standard test set library for MIP solvers. For these computations, we used up to 2048 cores of the HLRN II supercomputer.

1 Introduction

Branch-and-bound is a very general and widely used method to solve discrete optimization problems. An important class of problems which can be solved using this method are *mixed integer programs (MIP)*. The challenge of these problems is to find a feasible assignment to a set of decision variables which yields a mini-

Yuji Shinano · Timo Berthold · Stefan Heinz · Thorsten Koch
Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany
e-mail: {shinano, berthold, heinz, koch}@zib.de

Tobias Achterberg
ILOG, on IBM Deutschland GmbH, Ober-Eschbacher Str. 109, 61352 Bad Homburg v.d.H., Germany, e-mail: achterberg@de.ibm.com

* Supported by the DFG Research Center MATHEON *Mathematics for key technologies* in Berlin.

mum/maximum value with respect to a given linear objective function. The feasible region for these problems is described by linear inequalities. In addition a subset of the variables are only allowed to take integer values. These problems are *NP*-hard in general [12].

The well-known idea of branching is to successively subdivide the given problem instance into smaller problems until the individual subproblems (or sub-MIPs) are easy to solve. During the course of the algorithm, a branching tree is generated in which each node represents one of the subproblems. To be able to prune the vast majority of nodes at an early stage, sophisticated mathematical techniques are used. This allows a dramatic reduction of the size of the branching tree. Typically, problems with ten thousand variables and constraints (i.e., approximately 2^{10000} potential solutions) can be solved by investigating a few hundred thousand branch-and-bound nodes.

State-of-the-art MIP solvers such as CPLEX [3], Gurobi [1], or SCIP [8] are based on a branch-and-cut [15] procedure, a mathematically involved variant of branch-and-bound. Parallelizing branch-and-cut algorithms has been proven to be difficult, due to fact that the decisions involved depend on each other [16]. State-of-the-art codes *learn* from the decisions already taken, assuming a sequential ordering. Furthermore, basically all algorithmic improvements presented in the literature aim at reducing the size of the branching tree, thereby making a parallelization less effective and even more difficult. The latter is due to the observation, that they typically increase the need for communication and make the algorithm less predictable. Therefore, a well-designed dynamic load balancing mechanism is an essential part of the parallelizing branch-and-cut algorithms.

Since its introduction in 1992, the MIPLIB [11] has become a standard test set library used to compare the performance of MIP solvers. The MIPLIB contains a collection of difficult real-world instances mostly from industrial applications. Its availability has provided an important stimulus for researchers in this active area. The current version, MIPLIB2003 [9, 4], contains more than thirty unsolved instances when it was originally released. This number could be reduced to six; stalling at this level since 2007. These six instances resisted all attempts of the commercial vendors and the research community to solve them to proven optimality.

Algorithmic improvements for state-of-the-art sequential MIP solvers have been tremendous during the last two decades [10]. For an overview on large scale parallelization of MIP solvers, see [18]. Most of these approaches struggled, however, to catch up with the performance of state-of-the-art commercial and non-commercial sequential MIP solvers when it comes to solving really hard MIP instances of general nature. Many unsolved instances of MIPLIB2003 were first solved using sequential solvers [13].

In the following we describe how we developed a massive parallel distributed memory version of the MIP solver SCIP [5] to harness the power of the HLRN II supercomputer [2] in order to solve two of the remaining open instances of the MIPLIB 2003.

2 SCIP– Solving Constraint Integer Programs

SCIP (Solving Constraint Integer Programs) is a framework for constraint integer programming. Constraint integer programming is an extension of MIP and a special case of the general idea of *constraint programming (CP)*. The goal of SCIP is to combine the advantages and compensate the weaknesses of CP and MIP.

An important point for the efficiency of MIP and CP solving algorithms is the interaction between constraints. SCIP provides two main communication interfaces:

(i) propagation of the variables' domains as in CP and
(ii) the linear programming relaxation as in MIP.

SCIP uses a branch-and-bound scheme to solve constraint integer programs (see Section 2.2). The framework is currently one of the fastest MIP solvers [14], even so it is suitable for a much richer class of problems. For more details about SCIP we refer to [7, 8, 5].

2.1 Mixed integer programs

In this paper, we only focus on mixed integer programs (MIPs), which can be defined as follows:

Definition 1 (mixed integer program). Let $\hat{\mathbb{R}} := \mathbb{R} \cup \{\pm\infty\}$. Given a matrix $A \in \mathbb{R}^{m\times n}$, a right-hand-side vector $b \in \mathbb{R}^m$, an objective function vector $c \in \mathbb{R}^n$, a lower and an upper bound vector $l, u \in \hat{\mathbb{R}}^n$ and a subset $I \subseteq N = \{1,\ldots,n\}$, the corresponding *mixed integer program* $\text{MIP} = (A, b, c, l, u, I)$ is to solve

$$\begin{array}{lll} \min & c^T x & \\ \text{s.t.} & Ax \leq b & \\ & l \leq x \leq u & \\ & x_j \in \mathbb{R} & \text{for all } j \in N \setminus I \\ & x_j \in \mathbb{Z} & \text{for all } j \in I. \end{array}$$

The goal is to find an assignment to the (decision) variables x such that all linear constraints are satisfied and the objective function $c^T x$ is minimized. Note that, the above format is quite general. First, maximization problems can be transformed to minimization problems by multiplying the objective function coefficients by -1. Similarly, "$\geq$" constraints can be multiplied by -1 to obtain "$\leq$" constraints. Equations can be replaced by two opposite inequalities.

The *linear programming relaxation* is achieved by removing the integrality conditions. The solution of the relaxation provides a *lower bound* on the optimal solution value.

2.2 Branch-and-bound

One main technique to solve MIPs is the branch-and-bound procedure. The idea of *branching* is to successively subdivide the given problem instance into smaller subproblems until the individual subproblems are easy to solve. The best of all solutions found in the subproblems yields the global optimum. During the course of the algorithm, a *branching tree* is generated in which each node represents one of the subproblems.

The intention of *bounding* is to avoid a complete enumeration of all potential solutions of the initial problem, which usually are exponentially many. For a minimization problem, the main observation is that if a subproblem's lower (dual) bound is greater than the global upper (primal) bound, the subproblem can be pruned. Lower bounds are calculated with the help of the linear programming relaxation, which typically is easy to solve. Upper bounds are obtained by feasible solutions, found, e.g., if the solution of the relaxation is also feasible for the corresponding subproblem.

3 ParaSCIP

In this section, we introduce ParaSCIP, a parallel extension of SCIP. The design goals of ParaSCIP are to exploit SCIP's complete functionality, to keep the interface simple, and to scale to at least 10 000 cores in parallel.

We will focus on two important features, the dynamic load balancing and the checkpointing mechanism.

3.1 A dynamic load balancing mechanism

In this section we illustrate the workflow of the dynamic load balancing mechanism for ParaSCIP. Workload of a sub-MIP computation strongly depends on two factors. One is the number of branching nodes per solver, which may vary from one to several millions. The other is the computing time of a single branch-and-bound node, which may vary from less than one millisecond to several hours. Therefore, the dynamic load balancing mechanism is a key factor for the parallelization of branch-and-bound algorithms.

Initialization phase

In the beginning, the LOADCOORDINATOR, which acts as a master process, reads the instance data for a MIP model which we refer to as the *original instance*. This instance is presolved (see Section 4.2) directly inside the LOADCOORDINATOR.