A Parallel Macro Partitioning Framework for Solving Mixed Integer Programs

Mahdi Namazifar and Andrew J. Miller*

Industrial and Systems Engineering Department, University of Wisconsin - Madison,
1513 University Avenue, Madison, Wisconsin, USA
{namazifar,ajmiller5}@wisc.edu

Abstract. Mixed Integer Programs are a class of optimization problems which have a vast range of applications in engineering, business, science, health care, and other areas. For many applications, however, problems of realistic size can take a an impractical amount of time to solve on a single workstation. However, using parallel computing resources to solve MIP is difficult, as parallelizing the standard branch-and-bound framework presents an array of challenges. In this paper we present a novel framework called a Parallel Macro Partitioning (PMaP) framework for solving mixed integer programs in parallel. The framework exploit ideas from modern MIP heuristics to partition the problem at a high-level into MIP subproblems, each of which can be solved on a separate processor by an MIP algorithm. Initial computational resources suggest that PMaP has significant promise as a framework capable of bringing many processors to bear effectively on difficult problems.

Keywords: High-Performance Computing, Mixed Integer Programming, Primal Heuristics, Branch-and-Bound.

1 Introduction

In recent year great strides have been made in our ability to solve mixed integer programming (MIP) problems. Much of this progress has come through theoretical and algorithmic improvements that enable LP-based branch-and-bound and branch-and-cut algorithms to solve much large problems than previously possible. In spite of this progress, however, there are many MIP problems that remain practically intractable. For example, the MIPLIB library `http://miplib.zib.de/miplib2003.php` has numerous problems that take several days to solve; and, also, problem swhich have not yet been solved to optimality. Such facts encourage the use of high-performance computing resources for solving MIP's.

Parallelizing branch-and-bound has been studied in several papers (e.g., [4,8]). In general, two classic strategies for parallelizing branch-and-bound are node-based strategies, in which computations at the nodes of branch-and-bound tree are parallelized, and tree-based strategies, in which building and exploring the branch-and-bound tree is done in parallel [4]. In the latter approach, each processor is assigned a node of the tree and builds up the tree rooted at its assigned

* This research was supported by NSF grant CMMI 0521953.

L. Perron and M. Trick (Eds.): CPAIOR 2008, LNCS 5015, pp. 343–348, 2008.

node. The domain of the problem is thus split into several disjoint pieces, and each piece is processed by a single processor. Several solvers such as CPLEX [2], Xpress [7], and SYMPHONY [5], use such techniques to parallelize the branch-and-bound algorithm.

In this paper we propose a new framework that partitions the domain of MIP by using concepts derived from recently developed primal heuristics. Primal heuristics are methods which are used in the course of branch-and-bound to find good feasible solutions; examples include LP-and-fix, RINS, and local branching. By using primal heuristic methods both to define subproblems and to generate complementary cuts, we partition the feasible region at a high level into a set of subproblem MIPs that can be solved simultaneously in parallel.

The rest of the this paper is organized as follows: in section 2 we describe primal heuristics. In section 3 we sketch the whole view of the parallel MIP solver framework that we propose. Section 4 gives some notes about the implementation of the framework and some numerical results, and section 5 concludes the paper.

2 Primal Heuristics

Finding feasible solutions for mixed integer programs is important for at least two reasons: 1) for many applications identifying a good feasible solution is the primary goal of the MIP model; and 2) a feasible solution provides the branch-and-bound algorithm with an upper bound (considering a minimization problem) which accelerates the pruning process of a branch-and-bound algorithm.

Generally primal heuristics fall into two categories: construction heuristics and improvement heuristics. Construction heuristics try to produce a feasible solution from scratch. Improvement heuristics try to improve a given solution (or set of solutions) to find a better feasible solution. Here we briefly review some commonly used, powerful heuristics.

2.1 LP-and-FIX

This heuristic is very simple and is conceptually related to diving. Given a branch-and-bound tree node, the idea is to explore a sub-space defined by the current linear programming relaxation (LP) solution of the node. To do this we look at the LP solution and fix those integer variables that happen to take integral values in the relaxation solution. I.e., if the MIP problem is

$$(P) \qquad \begin{aligned} \min_{(x,y)\in\mathbb{R}^n\times\mathbb{R}^p} \; & cx + fy \\ \text{s.t. } & Ax + Gy \geq b \\ & x \geq 0, y \in \{0,1\}^p \, , \end{aligned}$$

let the the LP relaxation solution be $(\hat{x}, \hat{y})$. The LP-and-FIX problem is

$$(LP - FIX) \qquad \begin{aligned} \min_{(x,y)\in\mathbb{R}^n\times\mathbb{R}^p} \; & cx + fy \\ \text{s.t. } & Ax + Gy \geq b \\ & x \geq 0, y \in \{0,1\}^p \\ & y_j = \hat{y}_j \text{ for all } j \text{ with } \hat{y}_j \in \{1,0\} \, , \end{aligned}$$

2.2 Relaxation Induced Neighborhood Search (RINS)

Another highly useful primal heuristic is RINS [3], which can be seen as the improvement analog of LP-and-fix. Here we explore a sub-space defined by fixing those integer variables that take the same value in both the LP relaxation solution and a MIP feasible solution. Again letting $(\hat{x}, \hat{y})$ be and LP relaxation solution, and letting $(\bar{x}, \bar{y})$ is a MIP feasible solution for the problem. the RINS problem is

$$
\begin{aligned}
(RINS) \qquad & \min_{(x,y)\in\mathbb{R}^n\times\mathbb{R}^p} cx + fy \\
& \text{s.t. } Ax + Gy \geq b \\
& \quad x \geq 0, y \in \{0,1\}^p \\
& \quad y_j = \bar{y}_j \text{ for all } j \text{ with } \bar{y}_j = \hat{y}_j \ .
\end{aligned}
$$

2.3 Local Branching

Another type of improvement primal heuristic is Local Branching. In this heuristic we try to search a neighborhood around a MIP feasible solution. To do this search first an integer k is chosen. Then the neighborhood around a MIP feasible solution is defined as those y vectors that do not differ from the MIP feasible solution in more than k coordinates. As a result the local branching problem is defined as follows:

$$
\begin{aligned}
(LocalBranching) \qquad & \min_{(x,y)\in\mathbb{R}^n\times\mathbb{R}^p} cx + fy \\
& \text{s.t. } Ax + Gy \geq b \\
& \quad x \geq 0, y \in \{0,1\}^p \\
& \quad \sum_{j:\bar{y}_j=0} y_j + \sum_{j:\bar{y}_j=1} (1 - y_j) \leq k \ .
\end{aligned}
$$

3 The Parallel Macro Partitioning (PMaP) Framework

Here we explain the Parallel Macro Partitioning (PMaP) framework and its elements. We assume we have n processors, and we will discuss the work each processor does. We also have some data pools in the framework which can be considered as simple databases. We will briefly explain these pools in this section.

3.1 Processors

Brancher Processor. This processor generates subproblems and partitions the feasible region of the MIP. The brancher starts solving the MIP using branch-and-bound algorithm. At each node of the branch-and-bound tree, if there exists any MIP feasible solution in the pool of feasible solutions, the brancher generates a RINS problem and puts it in the subproblem pool. Then it adds the complement of the RINS cut (1) to the problem that it is solving:

$$
\sum_{j\in S^0} y_j + \sum_{j\in S^1} (1 - y_j) \geq 1 \tag{1}
$$

S^0: Set of variable indices with value of 0 for the variable in the feasible solution.
S^1: Set of variable indices with value of 1 for the variable in the feasible solution.

This cut guarantees that the part of the feasible space which is being processed in this RINS problem won't overlap with the problem which the brancher is solving. If there is no feasible solution in the feasible solution pool, the brancher generates a LP-and-FIX problem, puts it in the subproblem pool, and adds the complement of the LP-and-FIX cut

$$\sum_{j \in S^0} y_j + \sum_{j \in S^1} (1 - y_j) \geq 1 \tag{2}$$

S^0: Set of variable indices with value of 0 in the LP relaxation solution.
S^1: Set of variable indices with value of 1 in the LP relaxation solution.

The same process is done for Local Branching, and both new subproblems and complement cuts are generated at each node of branch-and-bound tree.

The primary purpose of the brancher is not to find feasible solutions, but rather to quickly create work for the worker processors. However, the brancher may well find feasible solutions during the branch-and-bound process. Whenever it finds a feasible solution, it writes it into the feasible solutions pool.

Worker Processors. They are $n-2$ processors that solve subproblems that are assigned to them using a branch-and-cut algorithm. During the solution process, they frequently check the feasible solution pool to see if any new solution has been found by other processors. If so, they update their cutoff value. Whenever they find a feasible solution they write it to the feasible solutions pool, and when they finish solving a subproblem, they send a message to the assigner processor to let it know that they are idle.

Assigner Processor. This processor looks for new subproblems in the feasible solutions pool. It also keeps track of the status of worker processors (busy or idle). As soon as a new subproblem appears in the subproblem pool, the assigner gets that and looks at the status of the workers. If there is an idle worker, the assigner gives the subproblem to the worker to solve. Otherwise, it waits until one of the workers declares that it is free.

3.2 Data Pools

Sub-problem Pool. All the subproblems generated by the brancher processor go to this pool. As was described earlier, the subproblems wait here until they are assigned to a worker processor to be solved.

Feasible Solution Pool. All the feasible solutions found by the brancher and worker processors are put here to be shared between all the processors. The brancher uses them to update it's cutoff value and generate subproblems. Workers use them to update their cutoff value.

4 Implementation and Numerical Results

4.1 Implementation

We have implemented PMaP using the free MIP solvers MINTO [6] and Coin-Cbc [1]. The brancher is implemented using MINTO and the workers are implemented using Coin-Cbc. The communication between the processors is done using either MPI (Message Passing Interface) or text files. The feasible solutions and subproblems pools are basically text files. We have currently implemented RINS and LP-and-FIX subproblems and cuts; we intend to implement local branching subproblems and cuts in the very near future. In general, PMaP is still at a preliminary stage, and we expect that we will be able to significantly improve its results in the very near future.

4.2 Results

Here we present the result of runs of PMaP on some of hard problems from MIPLIB 2003 (`http://miplib.zib.de/miplib2003.php`). As a benchmark, we use parallel CPLEX 10. The runs were performed on the Datastar machine in the San Diego Supercomputer Center (`http://www.sdsc.edu/us/resources/datastar/`). Because of the multi-threaded nature of parallel CPLEX, we could run it on at most 32 processors (maximum number of processors with shared memory) on Datastar. For fair comparison, we used the same number of processors for PMaP. Each instance ran for 30 minutes and Table 1 shows the best feasible solution each solver could find. As suggested by the table, the performance of PMaP is competitive with that of parallel CPLEX. This is significant, since serial CPLEX significantly outperforms serial Cbc on a single machine.

Table 1. The best feasible solution found by PMaP and parallel CPLEX 10 using 32 processors over 30 minutes

Problem	CPLEX	PMaP	Optimal Solution
glass4	1.60001e+09	1.65000e+09	1.20001e+09
markshare1	7	4	1
markshare2	25	16	1
portfold	-20	-21	-31
atlanta-ip	-	95.0098	90.0099
sp97ar	6.62541e+08	6.8753e+08	?
seymor	425	425	423
danoint	65.6667	65.6667	65.6667
dano3mip	698.6296	709.9629	?
swath	730.1	577.367659	467.407
net12	255	-	214

5 Conclusions

Our initial results suggest that PMaP is competitive with state-of-the-art commercial parallel softwares on the same architecture, even though PMaP itself is written entirely using open source code. Moreover, PMaP can, in principle, make use of hundreds or even thousands of processors simultaneously, and we intend to test its performance on truly massively parallel systems in the very near future. In addition, we expect that further development of PMaP will only enhance its performance. For example, the use of local branching subproblems and cuts may considerably enhance its performance. Moreover, it will likely be possible to use concepts of evolutionary methods (similarly to [9]) to quickly define even more subproblems that can simultaneously improve a upon a set of feasible solutions.

References

1. Coin-or project, `https://projects.coin-or.org/Cbc`
2. CPLEX. CPLEX User's Manual. CPLEX: a division of ILOG, Version 10 (2005)
3. Danna, E., Rothberg, E., Le Pape, C.: Exploring relaxation induced neighborhoods to improve MIP solutions. Mathematical Programming Series A 102, 71–90 (2005)
4. Gendron, B., Crainic, T.G.: Parallel brach-and-bound algorithms: Survey and synthesis. Operations Research 42, 1042–1066 (1994)
5. Ladányi, L., Ralphs, T.K., Trotter Jr., L.E.: Branch, cut, and price: Sequential and parallel. In: Jünger, M., Naddef, D. (eds.) Computational Combinatorial Optimization. LNCS, vol. 2241, pp. 223–260. Springer, Heidelberg (2001)
6. Nemhauser, G.L., Savelsbergh, M.W.P., Sigismondi, G.: Functional description of MINTO, a mixed integer optimizer. Operations Research Letters 15, 47–58 (1994)
7. Dash Optimization. Xpress optimizer (2008), `http://www.dashoptimization.com/`
8. Ralphs, T.K., Ladanyi, L., Saltzman, M.J.: Parallel branch, cut, and price for large-scale discrete optimization. Mathematical Programming 98, 253–280 (2003)
9. Rothberg, E.: Exploring relaxation induced neighborhoods to improve MIP solutions. INFORMS Journal on Computing 19, 1060–0189 (2007)