A Parallelization Mixing OR-Tools/Gecode Solvers on top of the Bobpp Framework

Tarek Menouer
PRISM Laboratory
University of Versailles Saint-Quentin-en-Yvelines
45 avenue des Etats-Unis
78035 Versailles, France
Email: Tarek.menouer@prism.uvsq.fr

Bertrand Le Cun
PRISM Laboratory
University of Versailles Saint-Quentin-en-Yvelines
45 avenue des Etats-Unis
78035 Versailles, France
Email: Bertrand.lecun@prism.uvsq.fr

Abstract—The novelty of this paper is to propose a parallel solver that mixes two Constraint Programming (CP) solvers, on top of our Bobpp framework, in order to have a parallel CP solver portfolio. The first chosen CP solver is OR-tools which is an open source sequential solver. We propose an external OR-Tools parallelization (partitioning/migration/load balancing) on top of the Bobpp framework, where different instances of the OR-Tools solver are executed on different cores. The second chosen CP solver is Gecode that is also an open source sequential and parallel solver. The Gecode parallelization is internal in a sense that the parallel strategy is included in the library. The advantage of Gecode is that it can be easily interfaced with other software. Bobpp is a parallel framework that provides an interface between: Combinatorial Optimization problems or CP solvers and the parallel machines. Bobpp is designed to propose several parallelizations according to the parallel programming environment and the architecture of a machine. The performances of our solver portfolio are illustrated by solving CP problems modeled in FlatZinc format.

Keywords—Parallelism; Constraint Programming; solver Portfolio.

I. INTRODUCTION

The innovations in hardware architectures as the multi-core parallel machines or MIC (Many Integrated Cores) and the progress in the field of parallelism and techniques used to resolve combinatorial problems using Constraint Programming (CP) solvers have achieved good performance. But it is never enough, nowadays we can not find a CP solver that can resolve all problems more effectively than other solvers, always there is a solver that performs better than others for some problems.

OR-Tools [24] is a library that includes a set of tools for Operational Research. It is developed in C++, the principle of this library is to explore the search space in order to find all possible solutions. OR-Tools is sequential CP solver. However, the OR-Tools library provides mechanisms called Monitor allowing some research control. Indeed, it is possible to save all the research branches, stop or replay a branch. Using OR-Tools Monitor we can do an external parallelization using the parallel Bobpp framework. In the literature there are several works that have been done to parallelize search spaces as studies used in Constraint Programming [8], [20], [21], [17], [25].

Gecode [22] is a CP solver, the main advantage of this solver is that it runs in parallel and it is easily interfaced with other systems.

To resolve the problem of competition between solvers, this paper presents a parallel CP solver portfolio mixing OR-Tools and Gecode solvers. As OR-Tools solver run in sequential, we present also the principle of method used to parallelize the OR-Tools search space using the parallel Bobpp framework [12], [16], this method is based on Work Stealing technique in order to partition the search-tree in a set of sub-trees and assigns each sub-tree to one computing core during the execution of the algorithm.

Section II presents briefly both CP solvers and the context of this work. Section III presents how to mix CP solvers on top of Bobpp framework. Section IV presents some experiences obtained with parallel CP solver portfolio. Finally, a conclusion and some perspectives are presented in section V.

II. CONSTRAINT PROGRAMMING SOLVERS

Constraint Programming solvers are very complex software systems. Each solver has its own characteristics, design, implementation and techniques to resolve combinatorial problems. According to the problem, choosing a particular solver is a difficult task which requires specialist knowledge about each solver.

In this work realised in the PAJERO OSEO-ISI project context, it has been proposed to solve combinatorial problems, using a portfolio based parallelization of several parallel CP solvers. This paper presents the results mixing the two CP solvers: OR-Tools and Gecode on top of our parallel Bobpp framework. The chosen CP solvers used to illustrate these principles are open source and are currently under active development.

OR-Tools [24] is an open source library developed in C++ developed by Google research team. The OR-Tools solver is mainly sequential. Several portfolio based parallelizations have been proposed but using local search method. OR-Tools does not propose a parallelization of its tree search algorithm. The OR-Tools parallelization using Bobpp use the monitor feature. With OR-tools monitor, we are able to externally record and stop a part of a search done in one solver, and replay this search in another OR-Tools solver to close the search on the sub-tree.
Gecode \cite{22} is also an open source solver. It is developed in C++. It supports the programming of new constraints and branching strategies. It can be easily interfaced to other systems. As Gecode is already parallel, we do not port it on top of Bobpp. However, we have interfaced it with our OR-tools/Bobpp solver. A parallel Gecode solver can be seen as a worker that helps to solve the problem. The principal is that threads share work (sub-trees) using a customize stack \cite{19}. To assure the load balancing between threads, when a thread completes the exploration of its sub-tree, it asks the manager of threads for another sub-tree to be explored.

III. MIXING OR-TOOLS/GECODE SOLVERS ON TOP OF THE PARALLEL BOBPP FRAMEWORK

Bobpp \cite{6}, \cite{14}, \cite{2}, \cite{12} is an open source parallel framework developed in C++ used to facilitate the development of sequential or parallel solver of Combinatorial Optimization problems. It provides several search algorithms classes, while being able to use different parallelization methods. Figure 1 shows a graphical representation of the interaction of application specific solvers, the methods and the parallel programming environments.

Bobpp has been mainly developed for Branch and Bound, Branch and Price and Branch and Cut methods which are methods used in mathematics programming but the search space of these methods is similar to the search space of algorithm used in the CP context. The parallelization of such tree search algorithms has been widely studied in the context of Divide and Conquer \cite{10}, \cite{4}, \cite{5}, Branch and Bound and all its variations \cite{7}, \cite{11}, \cite{3}, \cite{13}, \cite{1}.

A. Porting of OR-Tools Solver on top of Bobpp

The porting of OR-Tools on top of Bobpp has changed some of the main principles of Bobpp. Initially, Bobpp deals with search trees. The exploration consists in the scheduling of nodes that are stored in a global priority queue. At each step, each computing core gets one node from the priority queue, performs one level search i.e. generates the child nodes, and reinserts them in the priority queue. In this model, Bobpp manages all the nodes. The priority queue has the global knowledge of the search. With OR-Tools, the search is mainly performed by the OR-tools solver. Bobpp has no more global view on the search. It has been necessary to add a method to the priority queue, to test if the priority queue is empty, it means that there is not a node expecting to be explored and tests also if some cores are waiting for new inserted nodes.

To perform parallel search on a multi-core machine, each of the computing cores executes an instance of an OR-Tools solver. Furthermore to generate parallel tasks, we have to be able to stop the search on a specific node on a specific solver, then migrate the node from its origin solver to another one and then restart the search from the migrated node. As OR-Tools library provides a mechanism called Monitor allowing some control on the search, so in our solution, each solver use a specific OR-Tools monitor to store the path from the root node to a specific node in what we called OR-Tools-Bob-node.

The principle of this method is that the different cores of a machine share the work via a global priority queue. The search-tree is partitioned and allocated to the cores on demand. The threads perform the search locally and sequentially using the OR-Tools solver. As novelty used in the partitioning algorithm is the storage in advance of the right nodes during the exploration of a left branch. These right nodes may be good candidate to become OR-Tools-Bob-node saved in the global priority queue.

Figure 2 shows the progress of this method. When a thread finishes the search on its sub-tree, it gets function from the global priority queue. If the global priority queue is empty, the thread will declare itself as a pending thread. The other threads (working threads), which performing a search on their sub-trees, test if pending threads exist. In this case, when the working thread detect that exist at least one pending thread, the working thread stop the search on the left branch and creates an OR-Tools-Bob-node which represent the path from the root node until the right node, inserts the OR-Tools-Bob-node in the global priority queue and continues the search with the left node. The pending threads take effect by the insertion of a new OR-Tools-Bob-node in the priority queue. This method of partitioning is extensively studied in \cite{16}.

B. Parallel Constraint Programming Solver Portfolio

There are many different CP solving techniques and CP solvers. Each solver can solve some combinatorial problems according to the technique used for resolution. But we can not find a CP solver that solves all combinatorial problems, that is why in the last decade, the portfolio solver is used to
resolve the problem of competition between solvers and solve hard combinatorial problems quickly [15]. The objective of this work is to mix OR-Tools and Gecode solvers on top of Bobpp framework to have a parallel CP solver portfolio.

As the principle of Bobpp is based on the scheduling of nodes stored in a Bobpp global priority queue and it is a framework composed of basic classes that are extendible by the user. So to implement solver portfolio, it must start by inserting in the priority queue two Bob-nodes, one refers to OR-tools and the other refers to Gecode. Figure 3 shows an example of the using of the solver portfolio with four threads (two threads for each solver). At the beginning the OR-Tools thread 0 detects that an OR-Tools-Bob-node is in the priority queue, it takes it and starts the search, then it detects also that the OR-Tools thread 1 is pending, so the OR-Tools thread 0 stops the search on the left node, creates an OR-Tools-Bob-node, inserts it in the global priority queue and continues the search. The OR-Tools thread 1 takes effect by the insertion of new OR-Tools-Bob-node in the priority queue. It takes the node and performs the search.

In parallel, Gecode thread 0 is loaded and detects that is a Gecode-Bob-node in the priority queue. So it takes the Gecode-Bob-node that contains as information the number of threads asked by the user, initializes the second thread and it starts the communication between them. To stop the solver portfolio, the first solver which completes the exploration of the search space (found the optimal solution for Constraint Optimization problems or found all possible solutions or first solution for Constraint Satisfaction problem) stops the second solver.

IV. EXPERIMENTATIONS

To validate the approach used in this study, some experiments have been performed using a Linux computer that contains a bi-processor Intel Xeon X5650 (2.67 GHz) with the Hyper-Threading technology (12 physical cores) with 48 GB of RAM. The version used for OR-Tools is 2563 and for Gecode is 3.7.3. The solver portfolio is implemented without modifying the source code of OR-Tools and Gecode. For both solvers, tests are obtained with the basic version, that means without adding options for execution, all arguments are by default. All problems resolved are modeled using FlatZinc. FlatZinc is a low-level solver input language. It is designed to specify problems at the level of an interface to a Constraint Programming solvers [18]. In following, computation time presented in this section is given in seconds and it is an average of several runs.

Figure 4 and 5, show a comparison of performance between external parallelization using Bobpp framework for OR-Tools solver and an internal parallelization of the search algorithm used in Gecode solver. This comparison is based on the calculation of speed-up which refers to how much a parallel execution is faster than a corresponding sequential execution and efficiency that estimates how well utilized the cores for solving problems. This result is an average of several FlatZinc instances proposed in MiniZinc Challenge 2012. The aim of MiniZinc Challenge is to compare solvers and various constraint solving technologies on the same problem. We have used 40 instances (18 instance for Constraint Optimization Problems and 22 for Constraint Satisfaction Problems). As result with the both solvers we can obtain a speed-up using a parallel machine.

For both problems, Constraint Satisfaction and Optimization problems, figures 6 and 7 show the case where the OR-Tools performances is better than the Gecode performances. Figures 8 and 9 show the case where the Gecode performances is better than the OR-Tools performances.

As present in III-B, there is always the problem of competition between solvers, Figures 6, 7, 8 and 9 confirm
Tables I and II, shows results obtained with solver portfolio. For this experimentation we do an equitable partition between the number of threads. For example when we use 12 threads for solver portfolio, so we give for each solver 6 threads (6 threads for OR-Tools and 6 threads for Gecode). Result is evident, to any problem modeled using FlatZinc, parallel CP solver portfolio has the same performance as the performance of the best solver between OR-Tools and Gecode and the use of Bobpp framework does not increase the computation times.

V. CONCLUSION AND PERSPECTIVES

This paper presents a parallel solver portfolio on top of the Bobpp framework, that accommodates two parallel Constraint Programming solvers, parallel OR-Tools using Bobpp framework and parallel Gecode. As first perspective, to enrich and extend the set of problems resolved using Bobpp framework it is interesting to port on top of Bobpp a boolean satisfiability solver.
As second perspective, it would be good to give the user always the same solution for a specific problem, in sequential run is easy, it is the first solution but in parallel the first solution is not necessary. For this, partitioning algorithm used to parallelize OR-Tools solver will be adapted to give always the same solution if it is asked by the user.

The innovations in hardware architectures as the multi-core parallel machines have achieved good performance. But it is never enough, the number of cores in a parallel machine is limited. So as perspective, adapt the parallel CP solver portfolio with the distributed parallel machine and mixing another CP solvers as JaCoP [9] or CHOCO [23].

Bobpp achieves good speed-up on shared memory architectures. These results are obtained with several types of Combinatorial Optimization problems on different computers, Bobpp has also a parallelization on distributed memory machine mixed version MPI/Pthreads. To finish, perspective is to adapt the OR-Tools parallelization to the distributed parallelization of Bobpp.

REFERENCES