Experimenting with Constraint Programming on GPU

Fabio Tardivo
Department of Computer Science
New Mexico State University
Las Cruces, United States
ftardivo@nmsu.edu

The focus of my PhD thesis is on exploring parallel approaches to efficiently solve problems modeled by constraints and presenting a new proposal. Current solvers are very advanced; they are carefully designed to effectively manage the high-level problems' description and include refined strategies to avoid useless work. Despite this, finding a solution can take an unacceptable amount of time. Parallelization can mitigate this problem when the instance of the problem modeled is large, as it happens in real world problems. It is done by propagating constraints in parallel and concurrently exploring different parts of the search space. I am developing on a constraint solver that exploits the many cores available on Graphics Processing Units (GPU) to speed up the search.

1 Introduction and Background

1.1 Constraint programming

Constraint programming is a declarative programming paradigm, focused on problem modeling, instead of specifying the steps to find the solution.

A Constraint Satisfaction Problem (CSP) is a triple $P = (\mathscr{X}, \mathscr{D}, \mathscr{C})$ where $\mathscr{X}$ is a set of variables, $\mathscr{D}$ is the set of the variables' domains and $\mathscr{C}$ is a set of constraints. Formally a constraint specifies a subset of the cartesian product of the involved domains:

$$\begin{aligned}\mathscr{X} &= \{x_1, x_2\}\\ \mathscr{D} &= \{d_1, d_2\} = \{\{1,2,3,4,5\}, \{1,2,3,4,5,6,7,8,9,10\}\}\\ \mathscr{C} &= \{c_1, c_2\} = \{x_1 > 3, x_1 < x_2\} = \{\{4,5\}, \{(1,2),(1,3),\ldots,(5,10)\}\}\end{aligned}$$

A solution is an assignment of elements of their domains to variables that satisfy all the constraints.

A constraint solver is a procedure capable of returning the solution(s) to a CSP. It is build on three main operations:

Domains reduction The solver non-deterministically assigns a variable with a value in its domain.

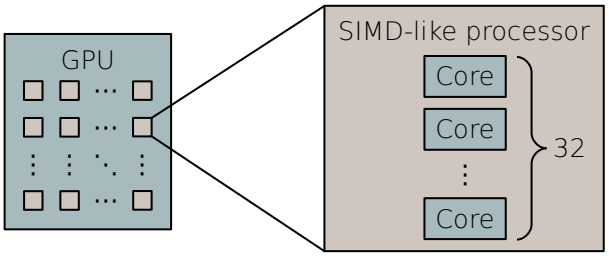
Constraints propagation The solver checks and possibly removes values that cannot occur in any solution.

Backtracking The solver restores the domains in case of a dead-end and records the solution in case of a success.

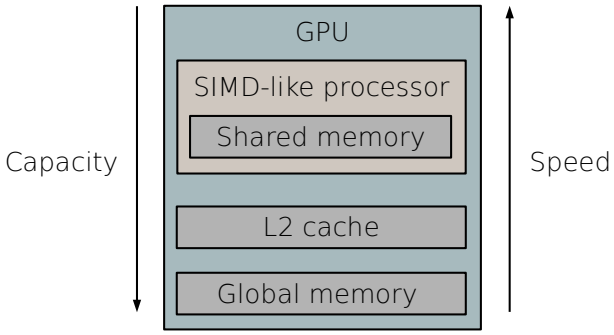
B. Bogaerts, E. Erdem, P. Fodor, A. Formisano,
G. Ianni, D. Inclezan, G. Vidal, A. Villanueva,
M. De Vos, F. Yang (Eds.): International
Conference on Logic Programming 2019 (ICLP'19).
EPTCS 306, 2019, pp. 427–432, doi:10.4204/EPTCS.306.58

1.2 CUDA

The increasing performance ratio between GPU and CPU encouraged researchers to use GPUs for non-graphics computations [10]. Computed Unified Device Architecture (CUDA) is an API to perform general computing on Nvidia GPUs.
From the CUDA point of view, the GPU is made of SIMD-like processors, each with 32 cores:



The memory hierarchy is mainly composed of 3 layers: Shared Memory, L2 Cache and Global Memory:



1.3 Parallel Constraint Programming

Over the years many approaches to parallel constraint solving have been explored. They are usually designed to run on computational clusters, and can be classified according to which part of the solving procedure is parallelized.

Parallel propagation There are two main approaches to parallelizing the constraint propagation. The first is to partition the variables among the computational units, duplicate the constraints and communicate the removed values [12]. The second is to partition the constraints among the computational nodes, duplicate variables and communicate the removed values [15].

Parallel search This approach can be generalized as a search space partition, where each part is assigned to a worker that acts as a standard solver. To address unbalanced workloads, the authors use centralized task dispatch [16], tasks pool [11] and tasks with priorities [5].

Portfolio method This method uses multiple solvers to find solutions. The solver configuration relay on a performance-problem database [13] or on the average performances with similar problems [1].

GPU This technique uses the GPU to accelerate the search process. The first implementation of this type of solver [3] only uses the GPU to propagate complex constraints. In subsequent applications, GPU has been used to perform local search heuristics like Adaptive Search [2] and Large Neighborhood Search [4]. GPU was also used for SAT problem [14], in detail the GPU was used to parallelize the unit-propagation and the exploration of the search tree's low part.

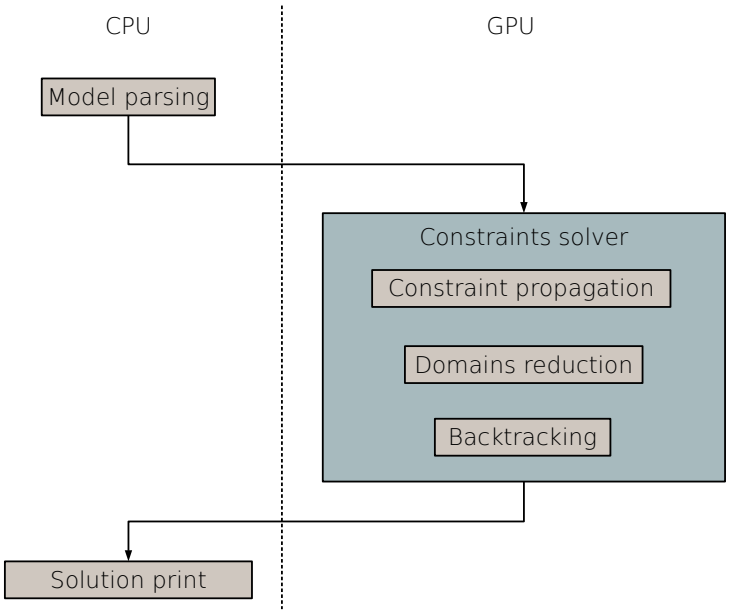
For a more complete and detailed survey, the reader can refer to [8, 7, 9].

2 CUBICS

In this section the solver named CUda BasIc Constraint Solver (CUBICS) is described. The first part gives an high level overview of the solver's workflow, the second part describes how constraint propagation and backtracking are parallelized and the last part is about some low level implementation details.

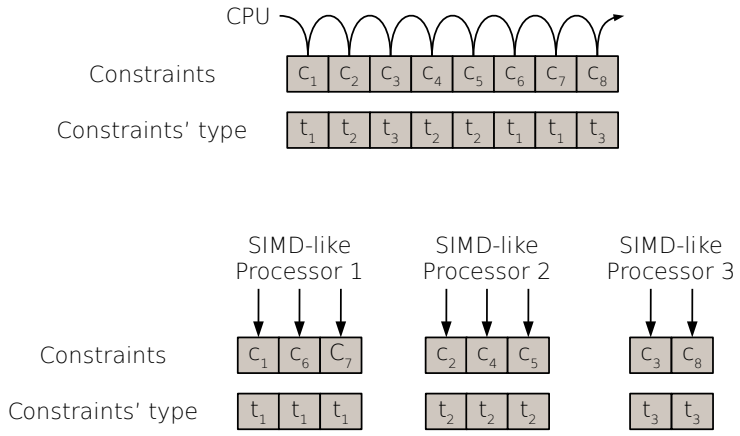
2.1 High-level design

The constraint solver performs the entire computation on GPU, since CPU-GPU data transfer can nullify the time gained by the parallelization. First, the CPU parses the problems, creates the data structures and moves them to the GPU's memory.
Control is then moved to the GPU, which performs labeling, constraint propagation, and backtracking. Here, constraint propagation is fully parallelized, while backtracking is partially parallelized and labeling is not parallelized at all, due to its sequential nature.
Once a solution is found, the control moves back to CPU which prints the result and potentially launches a new search.

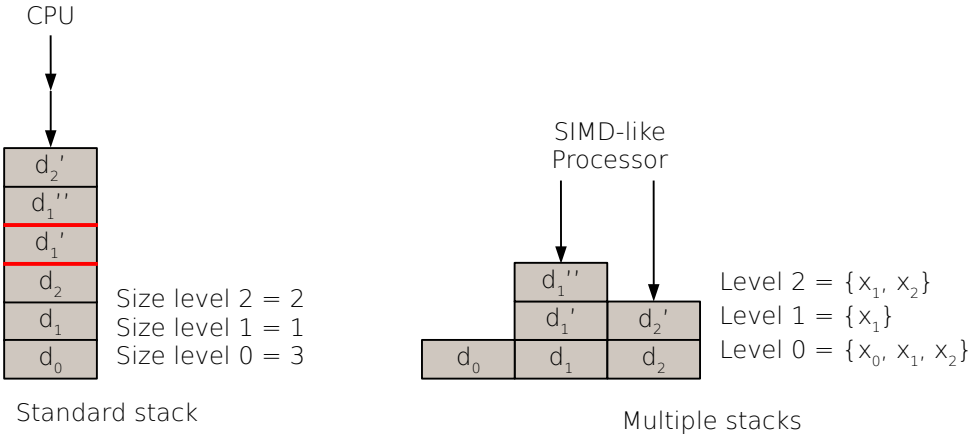


2.2 Parallelization

CUDA 5.0 introduced a new feature, referred to as Dynamic Parallelism. This feature allows a GPU thread to launch other GPU threads without returning the control to the CPU.
This mechanism is mainly exploited in CUBICS. Instead of propagating each constraint sequentially, the GPU propagates all of the constraints in parallel. The main GPU thread spawns other threads so that constraints of the same type are propagated by the same GPU processor. This is because each GPU processor behaves in a SIMD-like way, so it is desirable that each thread follows the same execution path.



Dynamic Parallelism is also used in backtracking to save and restore domains. This is done by moving from a single global stack to multiple stacks, one for each domain. In such configuration, the main GPU thread spawns as many threads as the domains to manage, without regard to the GPU processor they will be running on, since the code is common.



2.3 Implementation details

CUDA 6.0 introduced Unified Memory, which is a mechanism that abstracts the CPU and GPU memory, providing a unique memory address space. The data are initially stored in the CPU memory and implicitly moved on demand between the CPU and the GPU by the CUDA runtime system. This mechanism is used to facilitate both the creation of data structures and solutions retrieval.

One of the fundamental features of GPUs is the shared memory. It is an extremely fast on-chip memory, freely manageable by the programmer. This memory is exploited by the constraints propagation to cache the most accessed values. In case a constraint propagation needs to cache a significant amount of data, the solver reduces the number of threads per processor so the memory can accommodate the data.

3 Future works

There are many ways to extend the solver but few of them properly exploit the GPU architecture.
The first direction we will explore is the parallelization of the Large Neighborhood Search (LNS) methodology [6]. Once the solver finds a solution, it randomly resets some domains and performs another search looking for better solutions. This process can be parallelized creating many copies of the CSP, each with different reset domains. This approach explores much more search space at the same time, and allows for the sharing of the intermediate solutions so as to avoid looking for worse solutions, which increase the overall quality of the solutions. LNS has been successfully implemented on GPU in [4].
Another point that looks promising involves how the constraints are propagated. There are some global constraints for which propagation complexity is exponential, so the propagation algorithms for such constraints are polynomial approximations of the originals. With GPU, it will be possible to use more sophisticated algorithms that remove more values from the domains, improving the overall search time.

References

[1] Roberto Amadini, Maurizio Gabbrielli & Jacopo Mauro (2015): *SUNNY-CP: a sequential CP portfolio solver*. In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, ACM, pp. 1861–1867, doi:10.1145/2695664.2695741.

[2] Alejandro Arbelaez & Philippe Codognet (2014): *A GPU implementation of parallel constraint-based local search*. In: *Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on*, IEEE, pp. 648–655, doi:10.1109/PDP.2014.28.

[3] Federico Campeotto, Alessandro Dal Palu, Agostino Dovier, Ferdinando Fioretto & Enrico Pontelli (2014): *Exploring the use of GPUs in constraint solving*. In: *International Symposium on Practical Aspects of Declarative Languages*, Springer, pp. 152–167, doi:10.1007/3-540-49481-2_44.

[4] Federico Campeotto, Agostino Dovier, Ferdinando Fioretto & Enrico Pontelli (2014): *A GPU implementation of large neighborhood search for solving constraint optimization problems*. In: *Proceedings of the Twenty-first European Conference on Artificial Intelligence*, IOS Press, pp. 189–194, doi:10.3233/978-1-61499-419-0-189.

[5] Geoffrey Chu, Christian Schulte & Peter J. Stuckey (2009): *Confidence-Based Work Stealing in Parallel Constraint Programming*. In: *Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings*, pp. 226–241, doi:10.1007/978-3-642-04244-7.

[6] Jip J Dekker, Maria Garcia de la Banda, Andreas Schutt, Peter J Stuckey & Guido Tack (2018): *Solver-Independent Large Neighbourhood Search*. In: *International Conference on Principles and Practice of Constraint Programming*, Springer, pp. 81–98, doi:10.1007/978-3-642-40627-0_5.

[7] Agostino Dovier, Andrea Formisano & Enrico Pontelli (2018): *Parallel Answer Set Programming*. In Hamadi & Sais [9], pp. 237–282, doi:10.1007/978-3-319-63516-3_7.

[8] Ian P. Gent, Ian Miguel, Peter Nightingale, Ciaran McCreesh, Patrick Prosser, Neil C. A. Moore & Chris Unsworth (2018): *A review of literature on parallel constraint solving.* *TPLP* 18(5-6), pp. 725–758, doi:10.1017/S1471068418000340.

[9] Youssef Hamadi & Lakhdar Sais, editors (2018): *Handbook of Parallel Constraint Reasoning*. Springer, doi:10.1007/978-3-319-63516-3.

[10] E Scott Larsen & David McAllister (2001): *Fast matrix multiplies using graphics hardware*. In: *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, ACM, pp. 55–55, doi:10.1145/582034.582089.

[11] Laurent Michel, Andrew See & Pascal Van Hentenryck (2007): *Parallelizing Constraint Programs Transparently*. In: *Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings*, pp. 514–528, doi:10.1007/978-3-540-74970-7.

[12] Thang Nguyen & Yves Deville (1998): *A distributed arc-consistency algorithm*. *Science of Computer Programming* 30(1-2), pp. 227–250, doi:10.1016/S0167-6423(97)00012-9.

[13] Eoin O'Mahony, Emmanuel Hebrard, Alan Holland, Conor Nugent & Barry O'Sullivan (2008): *Using case-based reasoning in an algorithm portfolio for constraint solving*. In: *Irish conference on artificial intelligence and cognitive science*, pp. 210–216.

[14] Alessandro Dal Palù, Agostino Dovier, Andrea Formisano & Enrico Pontelli (2015): *CUD@SAT: SAT solving on GPUs.* *J. Exp. Theor. Artif. Intell.* 27(3), pp. 293–316, doi:10.1080/0952813X.2014.954274.

[15] Alvaro Ruiz-Andino, Lourdes Araujo, Fernando Sáenz & José J Ruz (1998): *Parallel Arc-Consistency for Functional Constraints.* In: *Implementation Technology for Programming Languages based on Logic*, pp. 86–100.

[16] Christian Schulte (2000): *Parallel search made simple*. In: *Proceedings of TRICS: Techniques foR Implementing Constraint programming Systems, a post-conference workshop of CP*, pp. 41–57.