Performance Analysis of Parallel Constraint-Based Local Search

Yves Caniou
JFLI, CNRS / NII, Japan
yves.caniou@ens-lyon.fr

Daniel Diaz
University of Paris 1-Sorbonne, France
daniel.diaz@univ-paris1.fr

Florian Richoux
JFLI, CNRS / University of Tokyo, Japan
richoux@jfli.itc.u-tokyo.ac.jp

Philippe Codognet
JFLI, CNRS / UPMC / University of Tokyo, Japan
codognet@jfli.itc.u-tokyo.ac.jp

Salvador Abreu
Universidade de Évora and CENTRIA FCT/UNL, Portugal
spa@di.uevora.pt

Abstract

We present a parallel implementation of a constraint-based local search algorithm and investigate its performance results for hard combinatorial optimization problems on two different platforms up to several hundreds of cores. On a variety of classical CSPs benchmarks, speedups are very good for a few tens of cores, and good up to a hundred cores. More challenging problems derived from real-life applications (Costas array) shows even better speedups, nearly optimal up to 256 cores.

Categories and Subject Descriptors G [*1.6*]: Constrained optimization; G [*2.1*]: Combinatorial algorithms; F [*2.2*]: Sorting and searching; D [*1.3*]: Parallel programming

General Terms Experimentation, Performance, Algorithms

Keywords combinatorial optimization, meta-heuristics, parallelism, implementation, Constraints, local search

1. Introduction

During the last decade, the family of Local Search methods and Metaheuristics has been quite successful in solving large real-life combinatorial problems [8–10]. Solving Constraint Satisfaction Problems (CSP) by Local Search is a way to tackle CSPs instances far beyond the reach of classical propagation-based solvers [3, 9, 11]. An effcient and generic domain-independent Local Search method named "Adaptive Search" was proposed in [3, 4]. It takes advantage of the structure of the problem to guide the search and can be applied to a large class of constraints (*e.g.,* linear and non-linear arithmetic constraints, symbolic constraints). Moreover, it intrinsically copes with over-constrained problems.

Parallel implementation of local search metaheuristics has been studied since the early 90's, when multiprocessor machines started to become widely available, see [12]. With the increasing availability of PC clusters in the early 2000's, this domain became active again [1, 5]. Apart from domain-decomposition methods and population-based method (such as genetic algorithms), [12] distinguishes between single-walk and multiple-walk methods for Local Search. Single-walk methods consist in using parallelism inside a single search process, *e.g.,* for parallelizing the exploration of the neighborhood. Multiple-walk methods consist in developing several concurrent explorations of the search space starting from different initial configurations, either independently or cooperatively with some communication between concurrent processes. Sophisticated cooperative strategies for multiple-walk methods can be devised but requires shared-memory or emulation of central memory in distributed clusters, impacting thus on performances. A key point is that independent multiple-walk methods are the most easy to implement on parallel computers without shared memory and can lead in theory to linear speed-up if solutions are uniformly distributed in the search space and if the method is able to diversify correctly [12].

We thus developed an independent multiple-walks of the Adaptive Search constraint solver (i.e. launching in parallel several search engines starting from different initial configurations and performing the computation in a purely independent manner) and benchmarked it with classical CSP benchmarks from the CSPLIB [7] and with a very difficult combinatorial problem, the Costas Array Problem (CAP). Historically, Costas arrays have been developed in the 1960's to compute a set of sonar and radar frequencies avoiding noise. The problem is to find an $n \times n$ grid containing n marks such that there is exactly one mark per row and per column and the $n(n-1)/2$ vectors between the marks are all different. It is convenient to formalize the CAP as a permutation problem on $\{1, 2, \ldots, n\}$ together with a set of constraints on 2D distance vectors. Although there are constructive methods to produce Costas arrays of order up to 27, it remains unknown if there exist any Costas arrays of size 32 or 33. Indeed, the solution density of Costas arrays is very low, *e.g.,* among the 27! permutations, there are only 204 Costas arrays. A very complete survey on Costas arrays can be found in [6]. Obviously, the search space for finding a Costas array of size n grows exponentially with n and it is thus a very good benchmark for search and combinatorial optimization methods.

PPoPP'12, February 25–29, 2012, New Orleans, Louisiana, USA.
ACM 978-1-4503-1160-1/12/02.

2. Parallel Performance Analysis

We performed our experiments on two different platforms:

- the Hitachi HA8000 supercomputer of the University of Tokyo with a total number of 15232 cores. This machine is composed of 952 nodes, each of which is composed of 4 AMD Opteron 8356 (Quad core, 2.3 GHz) with 32 GB of memory. Users can only have a maximum of 64 nodes (1,024 cores) in normal service and we used up to 256 cores in our experiments.
- the GRID'5000 infrastructure, the French national Grid for the research, which contains a maximum of 5934 cores deployed on 9 sites distributed in France. We used two subsets of the computing resources of the Sophia-Antipolis node: Suno, composed of 45 Dell PowerEdge R410 with 8 cores each, thus a total of 360 cores, and Helios, composed of 56 Sun Fire X4100 with 4 cores each, thus a total of 224 cores.

We first reported in [2] the performance of classical CSP problems from CSPLIB: `all-interval` (prob007), `perfect-square` (prob009), `magic-square`: (prob019). Table 1 shows that speedups are more or less equivalent on the HA8000 machine and on the GRID'5000 platform. They are good but tend to level after 128 cores, except for `perfect-square` on GRID'5000.

Platform	Problem	Time	Speedup on k cores			
		1 core	32	64	128	256
HA8000	MS 400	6282	20.6	31.7	41.3	54.1
	Perfect 5	42.7	29.5	44.6	49.1	57.0
	A-I 700	638	14.8	17.8	23.4	27.7
Suno	MS 400	5362	22.8	32.6	41.3	52.8
	Perfect 5	106	23	46.1	70.7	106
	A-I 700	662	15.8	19.9	23.9	28.3
Helios	MS 400	6565	20.6	31	44	-
	Perfect 5	139.7	24.5	46.6	77.2	-
	A-I 700	865.8	14.9	23.5	27.3	-

Table 1. Speedups on HA8000, Suno and Helios

Platform	Problem	Time on	Speedup on k cores		
		32 cores	64	128	256
HA8000	CAP21	160.4	1.96	4.16	10.0
	CAP22	501.2	2.01	3.90	8.24
Suno	CAP21	171	3.32	4.90	9.94
	CAP22	731	1.92	3.66	7.09
Helios	CAP21	153	1.51	4.17	-
	CAP22	1218	2.34	5.53	-

Table 2. Speedups on HA8000, Suno and Helios for large instances of CAP

For big instances of Costas arrays, speedups w.r.t. 1 core are nearly linear, *e.g.,* 107 for 128 cores and 218 for 256 cores for $n = 21$ on Suno. For $n = 22$, as sequential computation takes many hours, we limited our experiments to executions on 32 cores and above, see Table 2. We can observe that on all platforms, execution times are halved when the number of cores is doubled, thus achieving ideal speedup. This is graphically depicted in Figure 1 on a log-log scale. As a final result, we note that we can now solve $n = 22$ in about one minute on average with 256 cores on HA8000.

3. Conclusion and Future Work

We presented performances of a parallel implementation of a constraint-based local search algorithm, the "Adaptive Search" method in a multiple independent-walk manner. Each process is an independent search engine and there is no communication between the simultaneous computations except for completion. Performance evaluation on a variety of constraint satisfaction problems over two different parallel architectures (a supercomputer and a Grid platform) shows that the method is achieving good but not optimal speedups for classical benchmarks and presents linear speedups for the Costas Array Problem, a hard combinatorial problem.

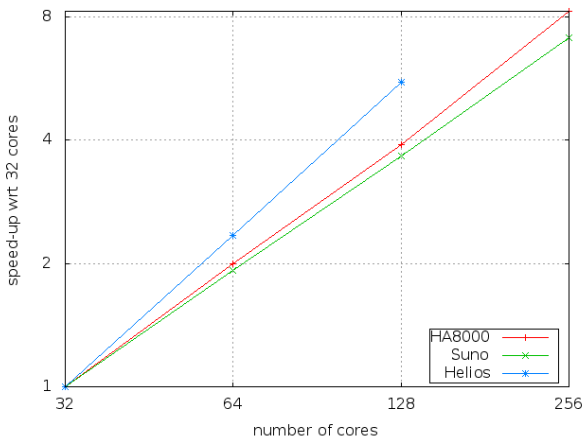

Figure 1. Speedups for CAP 22 w.r.t. 32 cores

Current work focuses on more complex parallel methods with inter-processes communication, i.e., in the dependent multiple-walk scheme, in order to further improve performance. The communication mechanism is being designed with the goals of (1) minimizing data transfers as much as possible, as we aim at massively parallel machines with no hierarchical memory, (2) re-using some common computations and/or recording previous interesting crossroads in the resolution, from which a restart can be operated.

References

[1] E. Alba. Special issue on new advances on parallel meta-heuristics for complex problems. *Journal of Heuristics*, 10(3):239–380, 2004.

[2] Y. Caniou, P. Codognet, D. Diaz, and S. Abreu. Experiments in parallel constraint-based local search. In *EvoCOP11, 11th European Conference on Evolutionary Computation in Combinatorial Optimisation*, LNCS 6622, pages 96–107. Springer Verlag, 2011.

[3] P. Codognet and D. Diaz. Yet another local search method for constraint solving. In *proceedings of SAGA'01*, pages 73–90. Springer Verlag, 2001.

[4] P. Codognet and D. Diaz. An efficient library for solving CSP with local search. In T. Ibaraki, editor, *MIC'03, 5th International Conference on Metaheuristics*, 2003.

[5] T. Crainic and M. Toulouse. Special issue on parallel meta-heuristics. *Journal of Heuristics*, 8(3):247–388, 2002.

[6] K. Drakakis. A review of costas arrays. *Journal of Applied Mathematics*, 2006:1–32, 2006.

[7] I. P. Gent and T. Walsh. CSPLIB: A benchmark library for constraints. In *proceedings of CP'99*, pages 480–481. Springer Verlag, 1999.

[8] T. Gonzalez, editor. *Handbook of Approximation Algorithms and Metaheuristics*. Chapman and Hall / CRC, 2007.

[9] P. V. Hentenryck and L. Michel. *Constraint-Based Local Search*. The MIT Press, 2005.

[10] T. Ibaraki, K. Nonobe, and M. Yagiura, editors. *Metaheuristics: Progress as Real Problem Solvers*. Springer Verlag, 2005.

[11] S. Kadioglu and M. Sellmann. Dialectic search. In *CP'09, Int. Conf. on Principles and Practice of Constraint Programming*. Springer Verlag, 2009.

[12] M. Verhoeven and E. Aarts. Parallel local search. *Journal of Heuristics*, 1(1):43–65, 1995.