Towards a Parallel Hierarchical Adaptive Solver Tool

Salvador Abreu, Danny Munera, Daniel Diaz

Towards a Parallel Hierarchical Adaptive Solver Tool

Salvador Abreu[1,2], Danny Munera[2], and Daniel Diaz[2]

[1] Universidade de Évora and CENTRIA, Portugal
salvador.abreu@acm.org
[2] University of Paris 1-Sorbonne - CRI, France
{danny.munera,daniel.diaz}@univ-paris1.fr

Abstract. Constraint satisfaction and combinatorial optimization problems, even when modeled with efficient metaheurisics such as local search remain computationally very intensive. Solvers stand to benefit significantly from execution on parallel systems, which are increasingly available. The architectural diversity and complexity of the latter means that these systems pose ever greater challenges in order to be effectively used, both from the point of view of the modeling effort and from that of the degree of coverage of the available computing resources. In this article we discuss impositions and design issues for a framework to make efficient use of various parallel architectures.

1 Introduction

Local Search techniques are used in combinatorial optimization and other situations where no tractable algorithm is known: this means that the problems they apply to are normally intrinsically difficult and tend to require massive computational resources when scaling in size. One way to deal with the complexity is to drive the search in parallel, a line which we have previously explored: see for instance [10,16].

We have been building on a metaheuristic technique called *Adaptive Search* [8] for which we developed a solver specialized for permutation problems. We resorted to independent multi-walks (IW) in order to extract parallelism from the Adaptive Search solver, on existing parallel hardware architectures, because of its relative simplicity. We have targeted the Cell/BE [2,10] and small to large scale HPC clusters using MPI [5,6]. In response to the observed limitations of IW at scale, we experimented with simple forms of inter-solver collaborative communication, which did not turn out as well as we hoped. This brought us to seek a more general framework for the coordination among solver instances. More recently, we have designed a highly parametric and customizable framework for the cooperative parallel execution of local search solvers. We decided to implement the framework in the X10 language [17], which we initially ran on clusters of NUMA machines using RDMA.

In this paper we explore candidate parallel target architectures and how to make efficient use thereof: we discuss design requirements and pitfalls which may impact performance.

In designing, implementing and assessing our framework, we also want to ensure that whatever convenience and flexibility we gain by using higher-level languages such as X10 [18] or Chapel[7], instead of libraries such as MPI, GPI [1] or POSIX Threads does not come at the cost of performance.

In this paper we start by presenting our framework for Cooperative Parallel Local Search in section 2, we then discuss various pertinent parallel hardware and software architectures in section 3, insofar as they will impact performance or condition the design of the local search solver. We conclude with a discussion, in section 4, of current and possible future lines of research for parallel local search.

2 A Framework for Cooperative Parallel Local Search

The straightforward way to use parallelism to improve the performance of Local Search is without any doubt the independent multi-walks strategy (IW). This approach takes advantage of parallelism by starting different isolated solver instances from different random initial points. Each solver explores a region of the search space, being totally unaware of the existence of the other solvers. While this parallel exploration of the search space proved very efficient for some problems its behavior tends to degrade when increasing the number of cores [5]. To overcome this state of things, we experimented with a *cooperative* approach by adding a communication mechanism to share information between solver instances. Sharing information is intended to improve the odds of getting to a good solution. There are thus many issues that must be addressed when designing a cooperative parallel search strategy for Local Search: in [19], the authors propose a list of important points for such cooperative solvers: *What information* is exchanged? *Between what processes* is it exchanged? *When* is the information exchanged? *How* is it exchanged? How is the *received data* used? Our previous experiments have confirmed the difficulty in making choices which result in better performance than IW [15,19], sometimes the impact is even detrimental! Clearly, this outcome may be explained by the overhead incurred in performing communication, but also by the uncertainty of the benefits stemming from abandoning the current candidate solution in favor of another, heuristics-based information which may or may not lead to an actual solution.

We came to the conclusion that a very parametric implementation is needed: maybe with good parameter tuning it will best IW. We thus recently proposed a general framework where different local search engines cooperate (through communication) in the quest for a solution [17]. The framework allows the user to define the trade-off between intensification and diversification, which is important to obtain the best performance: *intensification* holds the solvers within a promising part of the search space. In contrast, *diversification* helps in reaching remote parts of the search space [12]. The user can also customize the behavior of the framework thanks to a large set of parameters. Figure 1 shows an overview of the framework, where teams are shown to perform intensification internally, and "repel" each other to promote diversification.

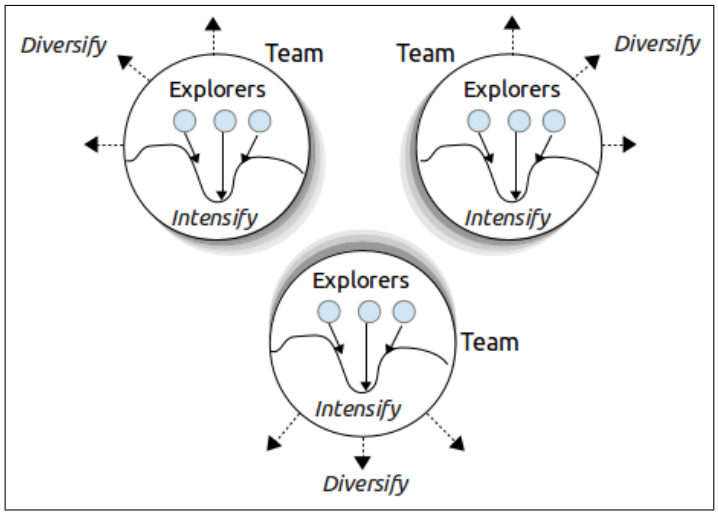

Fig. 1. Overview of the Cooperative Parallel Framework

The base cooperative search unit is the *Team* (see figure 2). Within a *Team* the communication strategy involves one *head* node and several *explorer* nodes. Periodically, each explorer sends information about its current state to the head node, in the case of the Adaptive Search method, this is the current $\langle \text{configuration}, \text{cost} \rangle$ pair. The head node manages an *elite pool* which stores a limited number of "best" configurations, as computed by all workers in the team. The elite pool may be managed according to several strategies, e.g. only accept into the elite pool new configurations whose cost is lower than the worst already present. Periodically, each explorer asks the head node for an "elite configuration". Here also, the head node may adopt different strategies in performing its selection: for instance it could pick the latest configuration, the "best" one, or a random one from the pool. Upon receiving the elite configuration, the explorer may, in turn, follow different strategies in deciding what to do about it: to ignore or switch to it. For instance, it can switch to the received configuration only if the cost is lower than the current one, yet perform this action only with a given probability. The progress that the current explorer will make on the newly adopted configuration is very likely to differ from that of the original node which produced it, because each node is following a different pseudo-random number sequence. In stochastic algorithms, this indeterminacy can give this configuration a "second chance," because it may be at the root of a new, different exploration path. Generally speaking, the explorer nodes within a team *intensify* the search towards the *local best* configuration, i.e. a promising neighbor in the search space. Note, however, that depending on the communication strategy, it may be possible for workers to "sit" on a good configuration for a while, before passing it on to the head node.

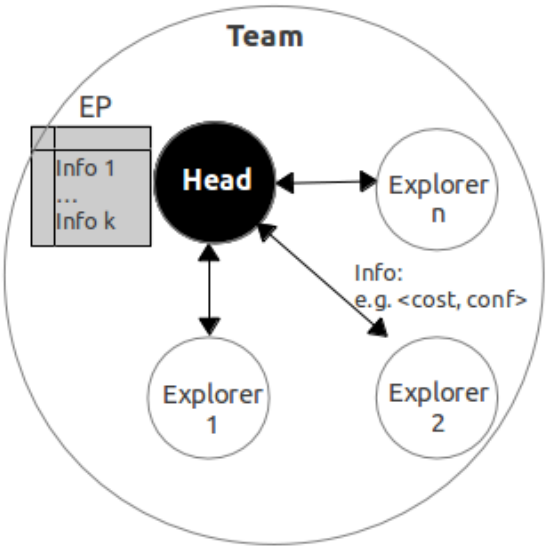

Fig. 2. Team Structure (Intensification)

In our framework, *Diversification* is ensured by the existence of different teams. Inter-team communication is used to make sure that distinct teams focus on different parts of the search space. In order to achieve this, the teams periodically share information about their current state with other teams. Once again, the information exchanged can be for instance a pair $\langle \text{configuration}, \text{cost} \rangle$, for the case where all teams execute the same algorithm (Adaptive Search, for instance). Several communication topologies are possible (e.g. a team can randomly select another team for the communication or a team can communicate to all others, ...), in line with the work presented in [15]. A team can then compute its *distance* to the received information, which could be for instance the number of the pairwise distinct values in the configuration vectors. In case two teams are too close, the "worse" one performs a *corrective action* in order to move away. The corrective action could be, for instance, a full restart of the algorithm of all (or just some) explorers in the culprit team.

The framework is very parametric and the user can control the size of the teams, the number of teams, the periodicity of the different communications, the size of the elite pool, the distance function, the corrective action, ... We instantiated this framework with our Adaptive Search method [8] for permutation problems using the X10 parallel language [18]. Preliminary results are very promising: the cooperative version already outperforms IW, especially in the harder cases. For more information the interested reader can consult [17].

3 Parallel and Distributed Architectures and their Idiosyncrasies

Modern computing systems, particularly high-performance ones, are increasingly parallel in several ways: within one processor at the instruction level, within one chip with multicore processors, at the system board level with multiple sockets, within one chassis with "compute accelerators" such as GPGPUs or manycore processors, at the local-area cluster level with multiple systems interconnected by a high-performance network. All of these form a hierarchy of parallel systems which needs to be taken into account when deploying multithreaded applications.

We now discuss some relevant parallel architectures and their specificities, how we expect to benefit from their use and what the main challenges in doing so are.

3.1 Multicore Systems

Multicore CPUs are now the norm in just about every computer being built. These are the heirs of traditional SMP multiprocessors of the 90s, but with a few twists of their own:

1. There are considerably more processors in today's multicore systems: it is not unusual to have 32-64 cores on a single motherboard.
2. Memory access is sensitive to the distance between the CPU the process is running and the actual physical memory bank its data is stored in.

Multicore systems are the "bread and butter" of modern architectures. In order to effectively exploit them, we must be able to generate enough paralellism to keep the available processors mostly busy. Communication among these processors may be safely regarded as lightweight, and as such need not be particularly avoided.

One difficulty which these systems bring to the table, and which must be dealt with on penalty of a drastic performance loss, is *non-uniform* memory access, or NUMA. The performance impact of doing remote memory access is typically in the 30-50% range [4], but may go up to 200% in pathological cases, i.e. careless access to memory may be up to three times slower than necessary. This means that it is important to control which processors accesses what regions of physical memory, in order to avoid doing non-local accesses.

3.2 Manycore Systems

Manycore systems are those which harbour many different processing units in a single package, with unified access to memory resources. Examples of *manycore* systems include the Intel MIC, or Xeon Phi, which in some ways resembles the IBM Cell/BE but at a larger scale.

By using manycore units, we expect to be able to exploit their similarity to regular SMP systems, which can mean that programs will tend to remain largely

unchanged. The programming model remains essentially the same: for instance, the Intel Xeon Phi resembles a normal Linux system, only with 260 processors.

Manycore systems such as Xeon Phi tend to have limited dedicated resources, especially RAM. Moreover, in order to extract good performance, one has to actually use all the available hardware threads.[1] Another issue which is difficult to deal with is concerned with the width of the datapaths: manycore processors have internal datapaths whose effective use requires the application to specify very wide operations, in the order of 128 to 512 bits per operation. These may require special measures as datatypes normally don't span such width.

3.3 GPGPUs

GPGPUs form a family of parallel systems, stemming from graphics rendering workflows, which is very well suited to vector arithmetic problems, at massive scale. These systems have the potential for a very high degree of parallelism – GPGPUs may have in the order of thousands of cores – which makes them appealing, as their cost gradually lowers.

These systems require programming which makes it very hard to reach good performance in processing complex data structures, as the good behavior of the GPU is tied to avoiding situations such as branch divergence[2] or non-coalesced[3] memory access. Even though the availability of manycore systems such as Xeon Phi has made it less urgent, mapping onto GPGPUs some of the parallelism found in solving local search problems remains an interesting proposal.

The most significant difference introduced by GPGPU programming in what concerns us, is that good performance requires execution to be done in a way which is very sensitive to uniform execution across threads.

3.4 HPC Clusters

A High-Performance Computing (HPC) cluster is a collection of mostly similar systems, connected by a high-speed low-latency network, usually Infiniband [13], these systems operate under unified management so as to appear as a single machine.

The networking architecture of Infiniband (IB) relies on the concept of Remote Direct Memory Access, or RDMA. This communication mechanism relies on establishing "channels" which map regions of virtual memory in the machines at both endpoints. These may be operated upon by "verbs" which tell the NICs to transfer data in either direction. The most interesting feature of this architecture is that machine-to-machine transfers are performed without any intervention by the main processors. One notable property of IB is its high

[1] This is tied to the fact that each core is single-issue and does not incorporate out-of-order execution.

[2] When not all threads in a thread group follow the same path, see [11].

[3] Accessing close parts of memory in a non-grouped way may severely impact performance, see [14].

bandwidth [4] and very low latency, partly because the IPC libraries eschew the normal TCP/IP stacks.

Usually one seeks to avoid doing communication in parallel search algorithms, as it quickly becomes a performance bottleneck. However, with the IB RDMA communication model, because IPC costs go down radically – both in terms of bandwidth and latency – it makes sense to pursue computational models which rely much more on collaboration among otherwise independent processes.

4 Opportunities and Threats

The outlook for the development of parallel solvers for local search and combinatorial optimization hinges on the effective use of parallel computing resources. We identified a number of these, each with idiosyncrasies which must be taken into account in order to benefit from the expectable performance gain.

- The diversity of communication relationships, which includes shared memory on the same NUMA node, non-local access to NUMA memory and remote access via RDMA over Infiniband, must dictate the placement of processes and the pairwise communication topology. One possible consequence is that knowledge of the hardware organization will favor some communication topologies, other than the centralized one which we have been experimenting with, as hinted to by the authors of [16], which delved into hierarchical multiprocessor topologies, shaping the communication topology to mimic the hardware.
- The elite pool ought to be kept on the head node's local memory.
- Manycore systems such as Xeon Phi pose specific challenges but provide interesting opportunities: the large scale at which they provide parallel processes which are on a very high-speed communication like – they operate on the same memory – likely means that processes which would otherwise entail costly communication can now appear viable. This argument is pertinent for both the intensification and diversification roles.
- To a lesser degree, communication over Infiniband presents similar opportunities, as the exchange may be done relatively fast, yet with no CPU intervention: effective resource usage will mean overlapping computation with communication.
- While interesting for the degree of parallelism they may offer, GPGPUs remain hard to exploit for the class of problems we want to address. The work described in [3], which implements IW Adaptive Search on GPGPUs, shows that these may effectively be used for local search. However, the performance gain (speedup) appears to taper off as the number of thread grows. It can be interesting to combine GPGPU with regular parallel solvers, clustering the GPU engines as teams geared for intensification.
- It remains to be seen whether the use of higher-level tools which abstract away some of the parallel architecture, while beneficial for the ease of experimentation, incremental design and code re-use, do not have too detrimental

[4] At present, FDR 4× infiniband does 56GBps.

an impact on performance. We want to explore the using languages such as X10 [18], Chapel [7] or UPC [9].
- On the other hand, we will experiment with general-purpose libraries such as MPI or GPI2 [1] which make use of the high-performance features of the underlying hardware, including RDMA and the integration of manycore processors.
- In memory organizations which support the PGAS (Partitioned Global Address Space), it is crucial that the data structures map onto appropriatly shared addresses, in coordination with the system's hierarchical memory layout.

To sum things up, the range of available options is exciting, particularly so as the preliminary explorations we have already carried out present such clear signs of good behavior.

References

1. GPI V2 Library Documentation. 2014.
2. Salvador Abreu, Daniel Diaz, and Philippe Codognet. Parallel local search for solving constraint problems on the cell broadband engine (preliminary results). In Yves Deville and Christine Solnon, editors, *LSCS*, volume 5 of *EPTCS*, pages 97–111, 2009.
3. Alejandro Arbelaez and Philippe Codognet. A gpu implementation of parallel constraint-based local search. In *PDP*, pages 648–655. IEEE, 2014.
4. François Broquedis, Jérôme Clet-Ortega, Stéphanie Moreaud, Nathalie Furmento, Brice Goglin, Guillaume Mercier, Samuel Thibault, and Raymond Namyst. hwloc: A generic framework for managing hardware affinities in HPC applications. In *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, pages 180–186. IEEE, 2010.
5. Yves Caniou, Philippe Codognet, Daniel Diaz, and Salvador Abreu. Experiments in parallel constraint-based local search. In Peter Merz and Jin-Kao Hao, editors, *EvoCOP*, volume 6622 of *Lecture Notes in Computer Science*, pages 96–107. Springer, 2011.
6. Yves Caniou, Philippe Codognet, Florian Richoux, Daniel Diaz, and Salvador Abreu. Large-Scale Parallelism for Constraint-Based Local Search: The Costas Array Case Study. *Constraints*, (to appear), 2014.
7. Bradford L Chamberlain, David Callahan, and Hans P Zima. Parallel programmability and the chapel language. *International Journal of High Performance Computing Applications*, 21(3):291–312, 2007.
8. Philippe Codognet and Daniel Diaz. Yet another local search method for constraint solving. In Kathleen Steinhöfel, editor, *SAGA*, volume 2264 of *Lecture Notes in Computer Science*, pages 73–90. Springer, 2001.
9. UPC Consortium et al. Upc language specifications v1.2. 2005.
10. Daniel Diaz, Salvador Abreu, and Philippe Codognet. Targeting the cell broadband engine for constraint-based local search. *Concurrency and Computation: Practice and Experience*, 24(6):647–660, 2012.
11. Tianyi David Han and Tarek S Abdelrahman. Reducing branch divergence in gpu programs. In *Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units*, page 3. ACM, 2011.

12. Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations and Applications*. Elsevier / Morgan Kaufmann, 2004.
13. InfiniBand Trade Association and others. *InfiniBand Architecture Specification: Release 1.0*. InfiniBand Trade Association, 2000.
14. Byunghyun Jang, Dana Schaa, Perhaad Mistry, and David Kaeli. Exploiting memory access patterns to improve memory performance in data-parallel architectures. *Parallel and Distributed Systems, IEEE Transactions on*, 22(1):105–118, 2011.
15. Rui Machado, Salvador Abreu, and Daniel Diaz. Parallel Performance of Declarative Programming using a PGAS Model. In Kostis Sagonas and Gopal Gupta, editors, *Practical Aspects of Declarative Languages (PADL 2013)*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2013.
16. Rui Machado, Vasco Pedro, and Salvador Abreu. On the scalability of constraint programming on hierarchical multiprocessor systems. In *ICPP*, pages 530–535. IEEE, 2013.
17. Danny Munera, Daniel Diaz, Salvador Abreu, and Philippe Codognet. A Parametric Framework for Cooperative Parallel Local Search. In *The 14th European Conference on Evolutionary Computation in Combinatorial Optimisation*, Granada, Spain, 2014.
18. Vijay Saraswat, Bard Bloom, Igor Peshansky, Olivier Tardieu, and David Grove. X10 language specification - Version 2.3. Technical report, 2012.
19. Michel Toulouse, Teodor G. Crainic, and Michel Gendreau. Communication Issues in Designing Cooperative Multi-Thread Parallel Searches. In I.H. Osman and J.P. Kelly, editors, *Meta-Heuristics: Theory & Applications*, pages 501–522. Kluwer Academic Publishers, Norwell, MA., 1995.