Handling heterogeneity for efficient implementations A case study on sequence comparison

Denis Trystram†
† Univ. Grenoble-Alpes and Institut Universitaire de France
trystram@imag.fr

***Abstract*—The focus of this talk is to present and discuss scheduling strategies in the context of hybrid parallel multi-core platforms composed of multicores with accelerators (GPUs). We put emphasis on general purpose policies developed at the middleware level (by opposition to codes tuned by the expert users for specific applications). We survey several recent results in various situations (off-line/on-line scheduling, for both independent tasks and precedence relations). These results are illustrated by actual experiments on the comparison of biological sequences in large databases.**

I. Introduction

As it can be observed in the few last TOP500 rankings, the largest parallel and distributed platforms are composed of hundreds thousands cores and more, with a sustained PetaFlop/s performance (10^{15} floating point operations per second). Reaching good efficiency in such large scale supercomputers is becoming a major issue and a big challenge. The situation is expected to worsen even more at exascale and beyond [3]. The profusion of flops available will likely be underused due to a much higher complexity.

Hybrid architectures have a growing impact on performances in High Performance Computing (HPC) and particularly for large bio-computing applications. Such architectures often use accelerators such as GPUs (Graphical Processing Units) [11] that are grouped with multiple CPUs on the same chip sharing the same common memory. Obviously, the efficiency of the implementations of parallel applications greatly depends on the quality of the organization of the elementary instructions at the finest level. This optimization is done by the users who are specialists of their codes.

The next generation of HPC platforms will be increasingly heterogeneous [10]. First, the number of cores within a node will increase to reach several hundreds and second, the nodes will be more diverse than today: fast compute nodes, hybrid compute nodes mixing general purpose units with accelerators, I/O nodes, nodes specialized in analytics or fault-tolerant, etc.. The interconnect of such a huge number of nodes will also lead to more heterogeneity. Such a diversity will give the opportunity to the application developers to use new features. For instance, the comparison of biological sequences may be extended with interactive visualization, introduce humans in the loop or couple the code with data analytics. Using hybrid platforms would potentially lead to better performances through the use of more appropriate resources depending on the target computations, but it has a cost in term of code development and mai induce erratic behaviors.

We will show in this talk that it is possible to design efficient algorithms for dealing with heterogeneity, whatever its form, at the system level. Such high level optimizations should be more considered by the application designers. They are not contradictory with fine grain *ad hoc* optimizations done within the applications and some studies should be developed at the interface of applications and the batch scheduler.

II. Description of the problems

A. Application, Platform, Resource manager

We consider a multi-core parallel platform with m identical CPUs and k identical GPUs. It can be hierarchical (in this case, the interconnection network is a logical tree) or there is a specific underlying topology (like a multi-dimensional torus). For the sake of simplicity, we restrict this presentation to one level of hierarchy (which is equivalent to a complete interconnection network), this allows us to avoid all the problems linked with communications or congestion. Of course, from the practical point of view this is an important issue, which has a strong influence on performances. There are several ways to take topological constraints into account. They will be discussed briefly during the talk.

An application is composed of n tasks that are independent or linked by precedence relations. For instance,

/15 $31.00 © 2015 IEEE
DOI 10.1109/IPDPSW.2015.123

in the sequence comparison problem, the elementary tasks correspond to pairwise comparisons using classical dynamic programming. The granularity of the tasks can be larger: they can be themselves executed on several processors in parallel in the case where a group of sequences are compared to a part or the whole database. Each task has two processing times depending on which type of processor it is assigned to. Usually, both processing times of a task are known in advance (or at least can be rather well estimated for structured applications like sequence comparisons).

As the application developers are mainly looking for performance, the objective of a schedule is to minimize the maximum completion time of the last finishing task (called the *makespan*). Minimizing the makespan in an heterogeneous context corresponds to minimize the maximum between the makespan on the CPUs and the makespan on the GPUs. Other objectives may be considered like minimizing the *stretch* (maximum or average), but they will not be discussed here.

B. Specialized or generalized scheduling?

The common algorithm for comparing biological sequences has been introduced by Smith and Waterman [12]. It is based on classical dynamic programming (which is quadratic in both space and time in term of sequence lengths). Using GPUs or other accelerators for improving the execution of such structured algorithms has been extensively studied. Notice here that using a GPU is well-suited for SW since it mainly consists in matrix operations. Several variants have been proposed for parallelizing this algorithm, mainly for static (off-line) computations. It is worth noticing here that a good algorithm can be obtaining while mixing both classical processors with accelerators (a part of the computations done in the CPUs and the rest on GPUs) [1].

We are interested here in designing generic approaches for efficiently implementing parallel applications where the scheduling is not explicitly part of the application. This way, the code is portable and can be easily adapted to the next generation of platforms.

C. New computational paradigms

The expected pressure on I/Os for the new requirements on applications is not sustainable at exascale and call for new approaches. Instead of saving raw data to disks for post-processing afterwards, the *in-situ analytics* proposes to perform data processing as closely as possible to where and when they are produced [15]. This paradigm becomes a whole including the execution of the applications, the processing for the analysis of results and I/O movements. New adequate scheduling strategies should be developed. In the first hand, this problem can also be reduced to an allocation problem of extra asynchronous tasks to idle computing units. But in the second hand, embedding analytics in applications brings extra difficulties by making the application more heterogeneous and imposing more constraints on the required resources. Thus, the main point here is to develop efficient scheduling algorithms for dealing with heterogeneity without increasing too much the global computational cost (or even, reducing it).

III. Analysis of the problem

Scheduling tasks on parallel identical machines has been extensively studied [4]. The basic problem is NP-hard, however, many variants lead to practical algorithms that are included into actual systems (communication delays, uniform speeds, etc.). Such solutions are either purely heuristics or sometimes provide approximations with constant performance guaranties (this means that the worst case scenario is kept bounded from the optimal). We are looking for low cost algorithms that deliver good approximations of the solution, typically with a ratio 3/2 or 2 for the minimization of the makespan.

Scheduling on hybrid platforms is more complex than the classical problem of scheduling on uniform machines [6]. However, it is easier than scheduling with unrelated machines since there are only limited values for the processing times (one per type of processing units). More details on identical, uniform, related and dedicated machines can be found in [4]. Very few papers deal with generic scheduling approaches of the hybrid problem, and none of them considers precedences constraints with performance guaranties. As we will present in the next section, this problem has efficient solutions in most practical situations.

IV. Approximation algorithms

A. Using existing solutions

There exist several practical generic scheduling algorithms for hybrid platforms like HEFT (Heterogeneous Earliest Finishing Time) [13] . However, such heuristic algorithms do not provide performance guaranties and thus, may lead to very bad executions for some instances.

The first attempt to implement an application is to use simple algorithms like *list scheduling* [5], which provides bounded performance guaranty. In list scheduling, a computing resource is never idle if one of the available

tasks could be started on the resource at that time. This algorithmic scheme has been extensively used and many variants have been analyzed. Most of them lead to small constant approximation ratio (2-1/m for the case of m identical machines). However, the use of the same strategy in a hybrid system, leads to arbitrary large values of the worst case performance ratio, even when there are no precedence constraints. Consider simply one task with a very large speed-up on GPU, which is misplaced. Moreover, the same situation also holds if we introduce some priority (like sorting the tasks by non-increasing speed-ups on GPUs). This becomes even worse for the case with precedence relations.

Such analysis of the problem of scheduling in hybrid platforms leads to the conclusion that the crucial point is to obtain the right assignment, or at least a reasonable one.

B. A Generic Principle

We propose to address the problem by a two-phases algorithm, where we determine first an assignment of the tasks, followed by a local schedule on each type of resources (CPU or GPU). There are several ways for solving the assignment problem, but in any case the method should be sophisticated enough to avoid unbounded approximation in the second phase. It is possible to solve this problem by an exact algorithm (which can easily be expressed as an integer linear program). If the size of the instance is too large, we can solve the corresponding fractional Linear Program and then, round the solution, or use an adequate direct approximation based for instance on global load arguments.

The same idea of using two successive phases may be used for the on-line case. In [2], the authors considered the scheduling of sequential tasks in mixed CPU/GPU platforms where the tasks arrive on-line one after the other. They introduced two smart rules for determining an assignment. Their analysis is based again on Graham's list scheduling leading to a 4-approximation, with improved ratio in some specific situations. An interesting point here was to be able to also provide a lower bound for this problem.

This principle also holds for the case of precedence relations (off-line). The assignment here is more complicated. However, we were able recently to propose a good assignment in [9]. This solution was obtained by a Linear Program, which is rounded to a feasible solution. In the second phase, an extension of list scheduling has been done to generate a feasible schedule.

The 2-phases principle can be applied on more sophisticated algorithms than list scheduling. For instance, this idea has been used with a dual approximation technique [7] for an off-line version of scheduling independent sequential tasks [1], leading to an approximation algorithm with a ratio equal to $4/3 + \frac{1}{3k}$. Since the assignment to the GPUs is done by a costly dynamic programming algorithm, a fast relaxed version with a ratio 2 has been derived.

Finally, it can also be adapted for dealing with parallel tasks [4] and may be very relevant in case of malleable tasks [14] (here, the number of machines allotted to a task in not fixed as an input of the problem, and the assignment will also determine it).

V. Case study of biological sequence comparison

Most of the algorithms presented above have been used for implementing the problem of comparing biological sequences in large databases [8]. Various parallel algorithms have been proposed for solving this classical problem. This problem corresponds to schedule independent tasks whose durations can be determined *a priori*. Many experiments have been run on actual databases of various species (humans, mice, dogs). The main result was that all the tested variants of the 2-phases algorithms overperformed the reference algorithm HEFT. on many instances. The obtained solutions provided non straightforward assignments on a mix of both GPUs and GPUs.

VI. Some Concluding Remarks

In this talk, we surveyed various methods for scheduling efficiently a set of tasks on hybrid parallel platforms. We presented a generic 2-phases approach where the key point is to avoid misplaced tasks that may lead to arbitrary bad solutions (typically putting a task with a very large acceleration at the wrong place). Thus, we proposed an approach that first provides a selection of the tasks between the CPUs and the GPUs, then, schedules them on each type of resources. This way, as we have shown, it was possible obtain efficient implementations with (low) bounded performance guaranties. There are several ways to do both assignments and local scheduling, leading to several variants. We are currently implementing such algorithmic approaches into actual systems.

REFERENCES

[1] R. Bleuse, S. Kedad-Sidhoum, F. Monna and G. Mounié and D. Trystram, *Scheduling Independent Tasks on Multi-Cores with GPU Accelerators*, Concurrency and Computations: practice and experience, 2014.

[2] L. Chen, D. Ye and G. Zhang, *Online Scheduling of mixed CPU-GPU jobs*, Int. Journal Foundations of Computer Science, Vol. 25, no 6, 2014.

[3] J. Dongarra et al. *The international exascale software project roadmap*. International Journal of High Performance Computing Applications, 25(1), 2011.

[4] M. Drozdowski, *Scheduling for Parallel Processing*, Springer, 2009.

[5] M. Garey and R. Graham, *Bounds for multiprocessor scheduling with resource constraints*, SIAM journal of Computing, vol. 4. 1975.

[6] A. Gupta, S. Im, R. Krishnaswamy, B. Moseley and K. Pruhs. *Scheduling heterogeneous processors isn't as easy as you think*, Proceedings of SODA 2012, pp. 1242-1253, 2012.

[7] D. Hochbaum and D. Shmoys. *Using Dual Approximation Algorithms for Scheduling Problems: Theoretical and Practical Results*, J. of ACM vol.34, 1987.

[8] S. Kedad-Sidhoum, F. Mendonca, F. Monna, G. Mounié and D.Trystram. *Fast Biological Sequence comparison on Hybrid Platforms*, Proceedings of ICPP, Minneapolis, USA, 2014.

[9] S. Kedad-Sidhoum, F. Monna and D.Trystram. *Scheduling tasks with precedence constraints on hybrid multi-core machines*, Proceedings of HCW, Hyderabad, India, 2015.

[10] P. Kogge et al., *Exascale computing study: technology and challenges in achieving exascale systems*, DARPA report, 2008.

[11] W. Lee et al., *Debunking the 100x GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU*. Proceedings of ISCA, Seznec, Weiser and Ronen editors, ACM, 2010.

[12] T. Smith and M. Waterman, *Identification of common molecular sequences*, Journal of Molecular biology, vol. 147, no 1, 1981.

[13] H. Topcuoglu, S. Hariri and M. Wu, *Performance-effective and low-complexity task scheduling for heterogeneous computing*, IEEE TPDS; 13(3):260-274, 2002.

[14] D. Trystram. *Scheduling Parallel Applications using Malleable Tasks on Clusters*, Workshop on Scheduling and Communication, IPDPS, San Francisco, 2001.

[15] F. Zheng et al. *FlexIO: I/O middleware for Location-Flexible Scientific Data Analytics*, proceedings of IPDPS, Boston, 2013.