Parallel adaptive algorithms scheduled by work stealing

Thierry Gautier, Jean-Noël Quintin, Jean-Louis Roch
MOAIS - INRIA - INPG - UJF - CNRS
Grenoble

Abstract
The work-stealing is an efficient well-known way to schedule tasks on a platform without knowledge about the platform and the application. However, even if theoretical bounds have been established to guarantee the performance of well formed programs having potential huge degree of parallelism, this approach must be augmented in order to increase the performance or to even hope optimal algorithm through a wide range of parallel architecture. During this talk, we will present our approaches based on 1/ an automatic adaptation of the parallelism of the program to the physical available parallelism thanks to an original recursive coupling of an optimal sequential algorithm with an efficient parallel algorithm ; 2/ a better scheduling decision which takes into account architectural characteristic (hierarchy) to drive the work stealing requests.

Mots-clés : Scheduling, work-stealing, adaptation, coupling

1. Motivation
The large expense of multiprocessors systems (MPSoC, CMP/multicores, SMP, GPU) that are combined in a single computer node and at a larger scale in clusters and grids, commonly used to run multi-programmed workloads, motivate exploring parallel models, algorithms and scheduling policies. The objective is the design of algorithms that achieve provable performances by self-adaptation to the execution context : computational units ; memory hierarchy ; networks and processors loads.

When dealing with on-line scheduling, most works are currently related to work stealing, a decentralized thread scheduler : whenever a processor runs out of work, it steals work from a randomly chosen processor. Popularized with Cilk (Charles Leiserson, SuperTech Group, MIT) in the 1996's, work stealing achieves provably good performances both in theory and practice for recursive fine grain multi-threaded computations which encompasses standard vector computations (C++ STL-like) from recursive range partitioning. Besides, work stealing is well suited for multiprogrammed environments, where external load (possibly from other components in the application) affects the speed of each shared unit ; it appears as an oblivious technique to adapt the application parallelism to the available resources (Michael Bender at SUNY Stony Brook). Furthermore, at the operating system level, new schedulers are currently studied to adapt the number of resources allocated to each work stealing application, e.g. adaptively to resource consumption on multi-core systems (work of K. Agrawal, S. Sen, Y. He, C. Leiserson, SuperTech group at MIT) or on grids (J. Dongarra, ICL Lab U. Tennesse) in order to optimize global performance. Also, work stealing is currently integrated in the kernel of various languages dedicated to multi-threading and high performance computing : C++ library Threading Building Block (Intel, v2.0 7/2007) ; the DARPA-HPCS triplet : Fortress language (Sun, β1.0 4/2007), Chapel (Cray, 2007) and X10 (IBM, 2007) ; Cilk++ (CilkArts, 2008) and now Intel Cilk+ (2010).

2. Contribution
Work stealing is well suited when there exists a parallel algorithm that performs almost the same work – number of operations performed, including communications and cache misses – than the best sequential one while having a very small depth – typically if the depth is polylog of the work –. However, this is generally not the case : work and depth are antagonist criteria that often cannot simultaneously be
minimized. Then, in order to adapt to the available resources and to fit to data distribution, several variations on work stealing are considered by the community. The generic underlying approach consists in finding a good mix of various algorithms.

**Adaptation of the parallelism of the application**

Based on work-stealing, the first part of this talk introduces algorithmic schemes to manage adaptivity at the application level: the objective is to improve performance by decreasing overheads induced by parallelism, namely extra arithmetic operations and communications due to the parallelism management. The resulting methodology allows us to design parallel programs that could be scheduled such that the parallelism degree is automatically adapted to available resources.

To minimize the work while preserving the depth, we focus on the recursive on-line coupling of two distinct algorithms: the one sequential that enforces to minimize work overhead; the other one parallel that minimizes the depth on an unbound number of processors. Due to work stealing property, most of the work is be performed sequentially locally on each processor. Then the challenge is to achieve provable oblivious optimal performances, i.e. reaching (asymptotic) optimal parallel work with no reference to the architecture.

For several applications (parallel computations on arrays from the STL standard library, image computations, anytime computations with interactivity constraints), we exhibit such optimal or near-optimal algorithms based on the development of complementary adaptive schemes of computation.

**Control of work stealing decisions on hierarchical architecture**

The second part of this talk deals with integration of the architectural hierarchy knowledge in order drive work stealing requests to improve performance. To deal with memory hierarchy, a classical variation is to start the computation from a pre-distribution that may be further corrected by work stealing. This scheme is used in the parallel versions of the STL: MCSTL/STXXL (Roman Dementiev, Johannes Singler, Karlsruhe University) and STAPL (Lawrence Rauchwerger group at Texas A&M University) and KaSTL. Moreover, run-time tests may be used to select the best performing algorithm (e.g. installation benchmarks in STAPL).

In order to better used shared cache of multicore architecture, another variation to reduce cache misses is based on sequential cache efficient algorithms and a control of the scheduling decision to avoid parallel "breadth" execution.

In the case of large scale hierarchical architecture, assumptions under classical theoretical bounds of work-stealing are not valid anymore since the communication cost is not taken into account. The time of each steal attempt is theoretically bounded by the same constant although the communications cost is not uniform on the platform. Thus, work stealing algorithms are unable to adapt the potential parallelism described at application level to the real parallelism at runtime. We will outline an original hierarchical work stealing algorithm, called HWS, for which we have bounded the total (expected) execution time of an application.

Several experimental results of various algorithms on multicores cluster or grid will illustrate the talk.

---

1. with some guarantees about performance.
Bibliographie


