Branch and Bound in a Data Parallel Setting

Extended Abstract

Sven-Bodo Scholz
Heriot-Watt University
S.Scholz@hw.ac.uk

Abstract

This paper investigates how branch and bound algorithms can be implemented in a functional, data parallel setting. We identify a general programming pattern for such algorithms and we discuss compilation and runtime aspects when it comes to mapping the programming pattern into parallel code. We use the maximum clique problem in undirected graphs as a running example and we present first experiences in the context of SaC.

1. Introduction

Branch and bound algorithms (which, in the sequel, we will refer to as *BBA*s) play an important role for many combinatorial search and optimisation problems. Typically these problems are NP-complete and require, at least in principle, the inspection of a search tree of exponential size. The key idea of branch and bound algorithms is the identification of certain bounds that allow pruning the search tree. That way, the overall runtime in many real-world applications can be brought down to a level where useful results are feasible despite the NP-complete nature of the underlying problem.

Application areas for these algorithms are vast including many areas that gain importance in the context of *big data* such as bio-informatics, computational chemistry, or social network analytics. The wide range of applications combined with the desire to deal with ever increasing amounts of data creates a demand for attempts to scale these applications to many-core systems. However, the challenges of successfully parallelising BBAs are many-fold. While a first cut seems rather obvious, i.e., spawning several threads that investigate separate branches of the search tree, achieving a parallel performance that scales well is far from trivial: The search tree typically is not well balanced, it may not even be statically known. The effectiveness of the bounding process often depends on knowledge gained by previous search space exploration and it may differ depending on where in the search space the exploration happens.

For many application areas, there exists a large body of work which investigates the effectiveness of different BBAs for individual problem instances. Often the perceived best solutions depend not only on the executing hardware, whether the algorithm is executed sequentially or in parallel, but they also depend on the given data itself. Low-level implementations of these algorithms are tedious, error-prone and typically require a lot of fine-tuning to achieve reasonable runtime performance.

This appears to be a setting where a declarative approach might help, be it in the form of a DSL or in the form of special language constructs. This paper presents our results when looking at BBAs from a data parallel angle. While a data-parallel approach may at first glance seem counter-intuitive for this seemingly inherent task-parallel class of algorithms, it turns out that nested reductions (folds) are a very apt vehicle for formulating BBAs. They provide an easy way to conveniently specifying the need for branching and, at the same time, they enable a compilation into effectively executable parallel code.

The main challenge, as in the manual case discussed extensively in the literature, is an effective declaration of the bounding needs. To our surprise, it turns out that very few language mechanisms suffice to express the bounding needs elegantly. We discuss what these mechanisms are and we argue their versatility. Furthermore, we show that SaC already provides suitable mechanisms. We use a classical problem from graph theory as running example to present and contrast several different specifications in SaC. This allows us to obtain initial performance figures and to relate these to the implementation features used.

Acknowledgments

This work was supported in part by grant EP/L00058X/1 from the UK Engineering and Physical Sciences Research Council (EPSRC).