Science of Computer Programming 112 (2015) 195–226

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Parallel parsing made practical

Alessandro Barenghi *, Stefano Crespi Reghizzi [1], Dino Mandrioli, Federica Panella, Matteo Pradella [1]

Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano, Piazza Leonardo Da Vinci 32, 20133 Milano, Italy

ARTICLE INFO

Article history:
Received 16 October 2014
Received in revised form 28 August 2015
Accepted 12 September 2015
Available online 30 September 2015

Keywords:
Parallel parsing algorithms
Syntax analysis
Parallel parser
Operator precedence grammar

ABSTRACT

The property of *local parsability* allows to parse inputs through inspecting only a bounded-length string around the current token. This in turn enables the construction of a scalable, data-parallel parsing algorithm, which is presented in this work. Such an algorithm is easily amenable to be automatically generated via a parser generator tool, which was realized, and is also presented in the following. Furthermore, to complete the framework of a parallel input analysis, a parallel scanner can also combined with the parser. To prove the practicality of a parallel lexing and parsing approach, we report the results of the adaptation of JSON and Lua to a form fit for parallel parsing (i.e. an operator-precedence grammar) through simple grammar changes and scanning transformations. The approach is validated with performance figures from both high performance and embedded multicore platforms, obtained analyzing real-world inputs as a test-bench. The results show that our approach matches or dominates the performances of production-grade LR parsers in sequential execution, and achieves significant speedups and good scaling on multi-core machines. The work is concluded by a broad and critical survey of the past work on parallel parsing and future directions on the integration with semantic analysis and incremental parsing.

1. Introduction

The current evolution of computing platforms led to the increase in the number of computing cores being the only way to increase their performances, beyond the power and frequency wall. As a consequence, the only way to cope with the ever increasing amount of data to be processed is to fully exploit the exposed parallelism whenever possible.

Parsing algorithms are used in all sorts of applications, with web browsing, text processing, compilation, and natural language processing being some of the most common application domains. However, in addition to being ubiquitous, they are also a significant exception to the above trend towards the exploitation of parallel architectures: the current state of the art reports no practically fruitful effort in this direction, save for an ad-hoc work tackling the parsing of HTML5 [51]. Open literature offers several historical proposals of parallel parsing algorithms, which however had no follow-up, let alone application, despite the growing amount of data which need to be processed, which has grown to such a pressing need that

* Corresponding author at: Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Tel.: +39 02 2399 3476.
E-mail addresses: alessandro.barenghi@polimi.it (A. Barenghi), stefano.crespireghizzi@polimi.it (S. Crespi Reghizzi), dino.mandrioli@polimi.it (D. Mandrioli), federica.panella@polimi.it (F. Panella), matteo.pradella@polimi.it (M. Pradella).
[1] Also with CNR IEIIT.

http://dx.doi.org/10.1016/j.scico.2015.09.002
0167-6423/

hardware accelerators have been developed to tackle it [38]. We refer the reader to Section 7 for a fairly comprehensive and critical analysis of the literature on this topic.

The most likely reason for this lack of practical parallel parsers is the intrinsically sequential nature of the classical deterministic (LR and LL) algorithms. For instance, assume to parse the language $L = L_1^*$, where L_1 is $\{1a^nb^n \mid n \geq 1\} \cup \{0a^nb^{2n} \mid n \geq 1\}$ through a deterministic shift-reduce parallel algorithm. Intuitively, it would be natural to map the parsing of each substring candidate to belonging to L_1 into a separate computation, henceforth called *worker*, and then to collect the partial results to decide whether the global sentence belongs to L or not. However, since the substrings belonging to L_1 can be arbitrarily long, any random or fixed policy to split the input into substrings is likely to be far from optimal; it may even be impossible to determine whether *ab* or *abb* groups are to be reduced in case a substring contains no 0s or 1s.

To cope with these issues, two straightforward approaches were pursued in the literature: speculative computations [41] or pre-scanning [39]. The former approach non-deterministically (speculatively) performs the parsing computation for all the possible cases: in our example, it carries on two parsing processes depending on whether the current substring is of type $1a^nb^n$ or $0a^nb^{2n}$, and discards the results of the incorrect computation as soon as possible. This approach, despite being effective, is not quite efficient: the computational effort of the parsing can be doubled or more, depending on the degree of nondeterminism. The latter approach involves a first lightweight scanning of the input to determine the viable splitting points: in the aforementioned example it would look for occurrences of 0 and 1 and split the input right before them. However, this approach introduces an overhead for the preliminary scanning which could require a pre-scanning over the whole input string. Summing up, the first approach may require an computational overhead to cope with the nondeterminism which is potentially more significant than the benefits provided by the parallelism, while the latter implies an $\mathcal{O}(n)$ worst-case preprocessing which, in case of simple languages, may take as much as a sequential parsing process.

The key point to overcome these impasses was our renewed interest in the "old-fashioned" operator precedence grammars (OPGs) invented by R. Floyd in his pioneering work [21] which laid the foundations of deterministic bottom-up parsing. After their first application in compiler construction, such grammars have been abandoned due to the advent of more powerful grammars, in terms of generative power, namely LR grammars. LR grammars [34,35] enable efficient, sequential parsing and generate all deterministic languages whereas OPGs do not. Nevertheless, we maintain the generative power of OPGs is quite adequate to formalize most languages of practical interest (Sections 4 and 6 discuss this issue) and we note that, thanks to their simplicity – and the simplicity and efficiency of their parsing algorithms – they are still used in compiler construction [28], the most notable example being the GNU Compiler Collection `gcc`, which actually employs an OP parser to handle expression parsing in C.

In this paper, we exploit the *local parsing* properties of OPGs to construct a *non-speculative* parallel parser. Intuitively, a language is locally parsable if, by inspecting a substring of bounded length, an (e.g., bottom-up) algorithm can deterministically decide whether the substring contains the right-hand-side of a production and can unequivocally replace it with the corresponding left-hand side. Local parsability is the key property that enables data-parallel parsing of isolated parts of the input so that their partial results can be recombined in a global syntax tree without backtracking: in other words, all the isolated partial syntax trees of a valid text are *final*.

In the following, we will present a systematic approach to exploit the local parsability of OPGs, providing a publicly available parser generator tool, and the results of an experimental campaign highlighting the speed-ups achievable w.r.t. popular sequential parsers, such as those generated by GNU Bison [2]. We chose as practical test-benches for our approach the JavaScript Object Notation (JSON) data description language, which offers a real-world validation for large input files, and the Lua programming language, to gauge how a language far richer than a data description language performs with the parallel lexing and parsing. We show that the minor theoretical limitations in terms of generative power of OPGs do not significantly affect the applicability of the approach: the changes needed to adapt the original BNF of the source language to OPG constraints are obtained in an original way by augmenting, and parallelizing as well, the initial phase of lexical analysis (or scanning).

The paper is organized as follows: Section 2 provides the theoretical foundations of OPGs, and their local parsability property, together with a parallel parsing algorithm. Section 3 provides our methodology for parallel lexical analysis, while Section 4 describes how we adapted the JSON and Lua languages to OP-based parsing. Section 5 describes the architecture of our parser generator, and Section 6 presents the results of the benchmark campaign. Section 7 compares our approach to previous research on parallel parsers, while Section 8 presents our conclusions and possible further research directions.[2]

2. Parallel parsing theory for OP grammars

In this section we develop a theory supporting the parallel parsing of OPLs. After summarizing their basic definitions and properties, we prove that OP grammars and languages enjoy the local parsability property, which is the key to make parsing parallel. Subsequently, we revise the sequential parsing algorithm for OPGs so that it applies both to terminal substrings and to partially processed ones (sentential forms). In this way, we devise a parallel procedure consisting of two (or more)

[2] An early description of the theory of parallel parsing for OPGs has appeared in [7] but, to make the paper self-contained, it is resumed and refined here in Section 2. On the practical side, PAPAGENO is described in the tool paper [6].