Available online at www.sciencedirect.com
ScienceDirect
Computer Languages, Systems & Structures 35 (2009) 196–215
COMPUTER LANGUAGES, SYSTEMS & STRUCTURES
www.elsevier.com/locate/cl

Efficient reconfigurable embedded parsers

Christos Pavlatos[a], Alexandros C. Dimopoulos[a,*], Andrew Koulouris[a], Theodore Andronikos[b], Ioannis Panagopoulos[a], George Papakonstantinou[a]

[a]*Iroon Polytechneiou, School of Electrical and Computer Engineering, National Technical University of Athens, Zografou 15773, Athens, Greece*
[b]*Department of Informatics, Ionian University, Plateia Tsirigoti 7, Corfu 49100, Greece*

Received 13 February 2007; received in revised form 19 July 2007; accepted 21 August 2007

Abstract

This paper presents a highly efficient architecture for the hardware implementation of context-free grammar (CFG) parsers. Its efficiency stems from an innovative combinatorial circuit that implements the fundamental operation of Earley's parsing algorithm in time complexity $O(\log_2|G|)$, where $|G|$ is the size of the CFG. Using this hardware architecture in a template form, we have developed an automated synthesis tool that, given the specification of an arbitrary CFG, generates the HDL (Hardware Design Language) synthesizable source code of the hardware parser for the given grammar. The generated source has been simulated for validation, synthesized and tested on a Xilinx FPGA (Field Programmable Gate Array) board. Our method increases the performance by a factor of one to two orders of magnitude, compared to previous hardware implementations, depending on the size of the grammar and the input string length. The speedup, compared to the pure software implementation, varies from two orders of magnitude for toy-scale grammars to six orders of magnitude for large real life grammars. This makes it particularly appealing for applications where (syntactic) pattern recognition response is a crucial aspect.

Keywords: Syntactic pattern recognition; Reconfigurable parsers; Context-free grammars; FPGA

1. Introduction

Many pattern recognition problems require syntactic analysis. In some applications this comes as a direct result of the application nature itself, e.g. speech recognition systems. In others, like image analysis or biomedical signal analysis, this requirement arises indirectly from the method applied on the problem [1–3]. In the latter case, it is a common practice to formulate the rules governing the relations between the patterns (e.g. relative position of objects in an image analysis system, or sequence of patterns in a biomedical signal analysis) by a grammar. Thus, the algorithms used for the syntactic analysis can significantly affect the efficiency of the overall pattern recognition system. Context-free grammars (CFGs) combine expressive power and simplicity. They are powerful enough to describe the syntax of programming languages [1] (almost all programming languages are defined via CFGs) and simple enough to allow the construction of efficient parsing algorithms. Additionally, general context-free methods are exploited today in various application domains, such as speech recognition, natural or programming language processing and image analysis, syntactic filtering, artificial intelligence [2,4–9]. The large amount of data manipulated by those methods

* Corresponding author. Tel.: +30 2107721529.
E-mail address: alexdem@cslab.ece.ntua.gr (A.C. Dimopoulos).

1477-8424/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cl.2007.08.001

and today's requirement of conforming to very strict real-time constraints, turns the implementation of CFG-parsers that are optimized for performance more useful and imperative than ever. Two key factors can positively influence the CFG-parser's performance: (i) the parsing algorithm itself, and (ii) the chosen implementation, i.e., serial or parallel, software or hardware.

With respect to the first key factor, many modifications [10,11] and improvements via parallelization [12] have been proposed for the well known classical parsing algorithms by Cocke–Younger–Kasami (CYK) [13,14] and Earley [15]. The CYK algorithm requires the grammar to be in Chomsky Normal Form (CNF). Every CFG can be put in CNF, so, this requirement does not restrict the generality of the CYK algorithm. The transformation of a CFG G into an equivalent one in CNF cannot be carried out without a price: the size of the latter can be $O(|G|^2)$ [16–18], where $|G|$ is the size of the CFG. However, this transformation takes place only once during the precomputational level and the transformation time does not influence the performance of an algorithm. Moreover, the transformation time is actually insignificant compared to the other tasks of precomputational level. Nevertheless, if the generated grammar G' after the transformation is larger than G (can be at worst case of size $|G'| = |G|^2$), this fact really influence the performance, since it is grammar size ($|G'|$) depended. Moreover, the lack of grammar transformation in our approach, in conjunction with the proposed automation of the design process (see Section 4), limits the precomputational level time (see Fig. 3) only to that of hardware synthesis (see Fig. 3).

With respect to the second key factor, Chiang and Fu [12] and Cheng and Fu [19] have presented designs using VLSI arrays for the hardware implementation of the above parsing algorithms, while Ibbara et al. [20] and Ra et al. [21] presented software implementations running on parallel machines. The hardware oriented approach was reinvigorated by presenting implementations in reconfigurable FPGA boards of the CYK algorithm [22,23] and Earley's algorithm [24]. The architectures proposed so far either fail to fully exploit the available parallelization of the parsing algorithms or demand excessive storage. The software approach, for example, executes parts of the parsing algorithms sequentially and, thus, does not achieve the maximum possible speedup. On the other hand, existing hardware methodologies must overcome the complexity imposed by the operations of the parsing algorithms, something that leads to increased storage needs. In order to relax the hardware complexity most of the proposed architectures implement the CYK algorithm, whose basic operations are much simpler than those of Earley's. The first FPGA implementation of Earley's algorithm was given in [24]. The approach proposed in this paper, uses a combinatorial circuit for the fundamental operation of Earley's algorithm. In this manner, we achieve an increase in time by a factor of one to two orders of magnitude, compared to previous hardware implementations [24], depending on the size of the grammar and the input string length. The speedup, compared to the pure software implementation, varies from two orders of magnitude for toy-scale grammars to six orders of magnitude for large real life grammars. This speedup is important in embedded real-time systems for syntactic pattern recognition applications. The merits of our approach are the following:

- We propose a new parallel parsing architecture, which compared to the so far presented architectures, reduces the required number of processing elements by a factor of $(n+1)(n+2)/2n$, where n is the length of the input string, while maintaining a simple and elegant communication scheme (see Section 3). Furthermore, we define a combinational circuit $C_{\otimes}$ that implements the fundamental operation $\otimes$ of the parsing algorithm [12,15] in extremely short time (execution time comparable to propagation delay of a few logic gates). Using the minimal number of $C_{\otimes}$ circuits (see Section 3), we compute every cell of the aforementioned architecture in parallel. Thus, we achieve two levels of parallelism: the *local* (cell-level), which corresponds to the execution of $\otimes$, and the *global* (architecture-level), which corresponds to the tabular form of the algorithm. At the same time, the above parallel implementation of the operation abolishes the need for storing the grammar. This is possible because, via a system of binary equations, the grammar characteristics are incorporated into the combinatorial circuit (see Section 5.1).
- The proposed design enhances the performance of the fundamental operation $\otimes$ by reducing its complexity to $O(\log_2(|G|))$ time. Furthermore the overall parsing complexity is reduced to $O(n \log_2(n|G|))$. Taking into consideration the hardware nature of the implementation, the presented architecture achieves a speedup factor that varies from two orders of magnitude for toy-scale grammars to six orders of magnitude for large real life grammars compared to software approaches (see Section 5.3).
- Our implementation does not require the grammar to be ε-free, where ε is the empty string. Requiring a grammar to be ε-free is a significant restriction, because the number of rules of the equivalent ε-free grammar may be significantly increased. Additionally, by implementing Earley's algorithm no CNF restriction (in the form of the grammar) is imposed and, therefore, no transformation of the initial grammar G is required. Although, all grammar