Context-Free Grammar Parsing for High-Speed Network Applications in Reconfigurable Hardware *

James Moscola †
jmm5@arl.wustl.edu

Young H. Cho ‡
young@arl.wustl.edu

John W. Lockwood §
lockwood@arl.wustl.edu

Washington University in St. Louis
Dept. of Computer Science and Engineering
St. Louis, Missouri 63130

1. INTRODUCTION

As the Internet continues to increase in popularity, new and more powerful technologies are emerging. These technologies are integrated into network applications for processing data as it traverses across the Internet.

Currently, many network applications are designed to provide network security. Such technologies include spam filters, virus scanners, and network intrusion detection and prevention systems. Other applications include packet filters, content-based routing, and natural language processing. At the core of each of these systems resides a rule-based pattern matcher, capable of detecting strings and/or regular expressions.

In recent years, many researchers have developed pattern matching hardware architectures capable of keeping pace with increasing network speeds and rule sets. However, naive pattern matchers do not consider the context of a match in the data. Therefore, they are susceptible to false positive identification. On a high-speed network, even a small number of false positives can surmount to an unmanageable amount of data.

This work intends to illustrate how context-free grammars (CFG) can be utilized to increase the accuracy of pattern recognition. CFGs provide a higher level of expressiveness than both strings and regular expressions by defining the semantics of a pattern within the structure of its language. This semantic information can then be used to reduce the number of false positive pattern identifications.

In addition to improving on existing network applications, having the ability to process CFGs and add semantic information to a network flow may open the door to new applications that are not currently possible with simple pattern matching (e.g. on the fly source code compilation).

2. PREVIOUS WORK

In previous work, hardware-based CFG parsers were implemented using the Cocke-Younger-Kasami (CYK) algorithm [1, 2]. While these implementations do manage to decrease the $O(n^3)$ time complexity of the CYK algorithm down to $O(n^2)$, the space required by the algorithm remains unchanged at $O(n^2)$, where n is the length of the input string. Such a large space requirement makes the CYK algorithm unsuitable for network applications that must maintain parsing information for millions of network flows simultaneously.

Other previous work includes a hardware-based implementation of an Early parser [3]. Again, the space requirements for this table driven parsing algorithm make it unsuitable for network applications.

The goal of our work is to design and implement a generalized high-speed CFG parser capable of processing and maintaining the state of millions of simultaneous network flows. Additionally, the parser should be robust and capable of recovering from errors in the input data stream.

3. PARSER ARCHITECTURE

The main components of the proposed architecture consist of a tokenizer (i.e. a pattern matcher) that is generated from a token list, a parsing structure that is generated from the production list of a grammar, and an error detection and recovery unit. A high level block diagram of the architecture is shown in figure 1. As demonstrated in our previous work [4, 5][1], a compiler automatically generates the CFG hardware from a Lex and Yacc style specification. The structure of the grammar is determined using the First and Follow set algorithms for software predictive parser generation. The generated hardware is a highly pipelined and parallel engine that recognizes patterns and the semantics of streaming data.

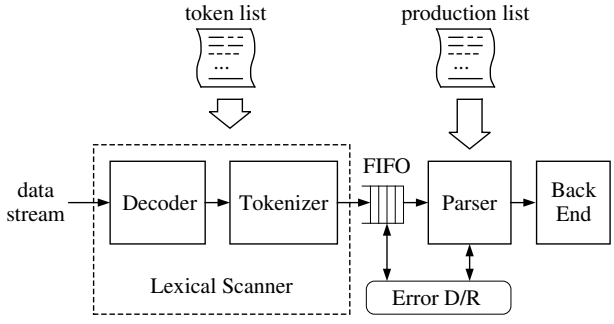


Figure 1: Hardware parser with error detection

*This research was sponsored by the Air Force Research Laboratory, Air Force Materiel Command, USAF, under Contract number MDA972-03-9-0001. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of AFRL or the U.S. Government.

†Expected graduation: May 2007

‡Advisor

§Advisor

[1]Supporting Papers: [4, 5]

4. TOKENIZER

In our previous work [5, 6], we illustrated several highly pipelined architectures capable of performing tokenization. The benefits of each architecture vary slightly. A regular expression chain is capable of matching regular expressions, but is not as compact as a pipelined character grid. The pipelined character grid is both compact and scalable, but cannot match full regular expressions. A hybrid architecture was also developed that encompasses the benefits of both of the previous architectures. The hybrid architecture is both scalable and capable of matching regular expressions while still maintaining the smaller size of the pipelined character grid.

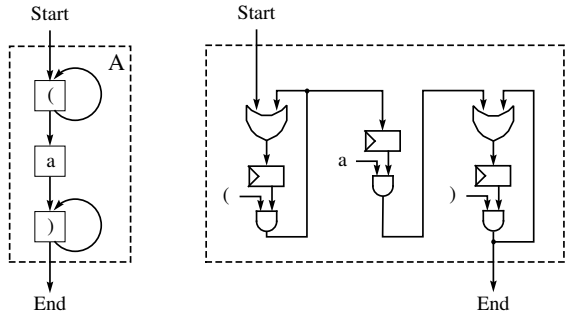
5. GRAMMAR PARSER

Unlike other parsers which use a table to look up the next state of the parser, we map the grammar rules directly onto a Field Programmable Gate Array (FPGA) in a highly pipelined structure. The structure of the grammar is determined using the First and Follow set algorithms for predicative parsers.

A simple example grammar is shown in figure 2. Figure 3a shows the finite-state automata required to match the grammar and figure 3b shows the logic required for the hardware parser. However, without a stack or some other method for keeping track of the nesting depth, this hardware is not a true CFG parser. As is, this hardware design will accept inputs that are not in the language specified by the grammar. For example, the invalid string "`( ( a ) ) )`" would be accepted by the hardware.

No.	Production
1	A → (A)
2	A → a

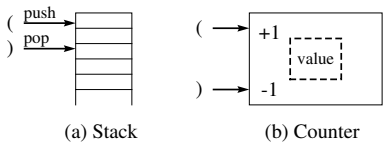
Figure 2: CFG for "a" with balanced parenthesis



(a) Finite-State Automata (b) Hardware Logic

Figure 3: Representations for grammar in figure 2

We intend to augment the hardware in figure 3b with a stack (figure 4a) to support true CFG parsing. However, to correctly parse millions of network flows simultaneously, each flow would require its own stack. Maintaining millions of stacks on-chip is not currently possible, and swapping a stack in and out of off-chip memory between network packets would dramatically decrease throughput. Thus, adding a stack would bring us further from our goal of developing a hardware parser capable of processing millions of network flows.



(a) Stack (b) Counter

Figure 4: Maintaining nesting depth with stacks and counters

In lieu of adding a stack for each network flow we want to parse, we can minimize the storage space required by employing small counters (figure 4b) for each of the tokens that can be nested. While millions of these counters still could not be maintained on-chip, the time required to write/read these counters to/from off-chip memory would be much less than that of a stack. However, both the size and number of counters need to be limited in order to minimize the amount of data that is swapped between network flows.

While stacks and counters may be able to maintain the state of a CFG parser during parsing, we intend to explore the possibilities of building parse trees after parsing. Using the hardware in figure 3b to output a series of tokens and production numbers, a back end software process can then build the parse tree and determine the context of each token. This software process can either be executed off-chip or on-chip using an embedded processor core.

6. REFERENCES

[1] Cristian Ciressan, Eduardo Sanchez, Martin Rajman, and Jean-Cedric Chappelier, "An FPGA-Based Coprocessor for the Parsing of Context-Free Grammars," in *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Napa, CA, Apr. 2000.

[2] Ciressan Cristian-Raul, Sanchez Eduardo, and Rajman Martin, "An FPGA-Based Syntactic Parser for Real-Life Unrestricted Context-Free Grammars," Technical Report No. 01/373 01/373, Swiss Federal Institute of Technology (EPFL), Lausanne (Switzerland), October 2001.

[3] Andreas Koulouris, Nectarios Koziris, Theodore Andronokos, George Papakonstantinou, and Panayotis Tsanakas, "A Parallel Parsing VLSI Architecture for Arbitrary Context Free Grammars," in *Proceedings of International Conference on Parallel and Distributed Systems (ICPADS)*, Tainan, Taiwan, Dec. 1998.

[4] Young H. Cho, James Moscola, and John W. Lockwood, "Context-Free Grammar based Token Tagger in Reconfigurable Devices," in *Proceedings of International Conference of Data Engineering (ICDE/SeNS)*, Atlanta, GA, USA, Apr. 2005.

[5] James Moscola, Young H. Cho, and John W. Lockwood, "Reconfigurable Context-Free Grammar based Data Processing Hardware with Error Recovery," in *Proceedings of International Parallel & Distributed Processing Symposium (IPDPS/RAW)*, Rhodes Island, Greece, Apr. 2006.

[6] James Moscola, Young H. Cho, and John W. Lockwood, "Fast Semantic based Identification of Regular Expressions using Reconfigurable Devices," in *submitted to IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Napa, CA, Apr. 2006.