An FPGA-based syntactic parser for real-life almost unrestricted context-free grammars

Cristian Ciressan[1], Eduardo Sanchez[2], Martin Rajman[1], and Jean-Cédric Chappelier[1]

[1] Computer Science Dept., LIA, CH-1015 Lausanne, Switzerland
[2] Computer Science Dept., LSL, CH-1015 Lausanne, Switzerland
Swiss Federal Institute of Technology

Abstract. This paper presents an FPGA-based implementation of a syntactic parser that can process languages generated by almost unrestricted real-life context-free grammars (CFGs). More precisely, we study the advantages offered by a hardware implementation of a parallel version of an item-based tabular parsing algorithm adapted for word lattice parsing. A description of the parsing algorithm and of the associated hardware design is provided. A method called *tiling*, that allows a better processor and I/O bandwidth exploitation is introduced. Finally, an evaluation of the design performance on real-life data is given and the measured 244 speed-up factor makes our design a promising solution for Natural Language Processing applications, for which parsing speed is an important issue.

1 Introduction

Efficient parsing is an important element in most NLP applications, especially for those with strong real-time and/or data-size constraints. Typical examples of such applications include advanced Information Retrieval [7, 8], Text Mining [5, 6], and the design of vocal interfaces. In such cases parsing can either enhance performance by integrating more syntactic knowledge about the language or allow a better integration of syntactic processing with speech recognition.

As far as context-free languages are concerned, VLSI implementations, based on 2D-array of processors have been proposed, both for the CYK [2] and the Earley [3] parsing algorithms. Although these designs meet the usual VLSI requirements (constant-time processor operations, regular communication geometry, uniform data movement), the hardware resources they require do not allow them to accommodate unrestricted real-life context-free grammars used in large-scale NLP applications[1]. In previous contributions [11, 12], we have proposed two FPGA-based implementations of the CYK algorithm [4] that can process real-life CFG. We choose the FPGA technology because it provides hardware designers with flexible and rapid system prototyping tools and also because, in the near future, it is expected to find FPGA components in general-purpose processors.

[1] for instance, the CFG extracted from the SUSANNE corpus [1] used in our experiments contains 1,912 non-terminals and 66,123 grammar rules.

The main advantages of our previous designs were: (1) their ability to parse word lattices, which makes them better adapted for integration within speech-recognition systems, (2) their scalability, and (3) a 30 speed-up factor [12] when compared against a software implementation of the CYK algorithm. On the other hand, the weak points of the these designs were: (1) the fact they require CFGs written in Chomsky Normal Form (CNF), (2) a low average processor load and (3) a low average I/O bandwidth exploitation. These different drawbacks are addressed in this paper, and a new implementation is proposed. The new design is based on a general item-based chart parsing algorithm (an enhanced version of the CYK algorithm [10, 13]) that allows the processing of almost unrestricted CFGs. A better load of the processors and exploitation of the I/O bandwidth is obtained with a method called tiling. Among other improvements on the previous designs, we mention a simpler initialization procedure, and a faster processor implementation.

2 The general item-based chart parsing algorithm for word lattice parsing

The implemented parsing algorithm is designed to process *non partially lexicalized* CFGs, a subclass of CFGs (henceforth denoted as nplCFG) in which terminal symbols, i.e. words, only occur in rules of the form $X \rightarrow w_1 w_2 ... w_n$, (called lexical rules). For our design, the restriction to nplCFGs is very useful as it allows to restrict the processing of lexical rules to the initialization step only, which takes place outside the FPGA chip in the on-board general-purpose processor. In addition, to further increase the efficiency of the hardware implementation, we restrict to nplCFGs without chain rules (though this is not a restriction of the general item-based chart parsing algorithm, it strongly simplifies the FPGA design).

As the implemented algorithm is a tabular item-based parsing algorithm, the syntactic trees associated with an input sequence $w_1 \dots w_n$ are represented in the form of sets of elements called items stored in a table (henceforth called the chart). More precisely, if $w_{ij} = w_i w_{i+1} \dots w_{i+j-1}$ denotes the subsequence of $w_1 \dots w_n$, of length j and starting with w_i, the cell at row j and column i in the chart, contains the following two sets of items:

- a set $N1_{i,j}$ of all non-terminals X that can derive w_{ij}, defined as: $N1_{i,j} = \{X \in N : X \Rightarrow^\star w_{ij}\}$[2]
- a set $N2_{i,j}$ defined as: $N2_{i,j} = \{\alpha\bullet : \exists \beta \in N^+, \exists (Z \rightarrow \alpha\beta) \in R, \text{ s.t. } (\alpha \Rightarrow^\star w_{ij})\}$[3] which is the set of all sequences of non-terminals that can derive w_{ij}.

[2] Where N is the set of non-terminals of the grammar and $\alpha \Rightarrow^\star \beta$ indicates that the sequence β can be obtained from the sequence α by applying a finite number of rewriting rules.

[3] Where R is the set of the rewriting rules contained in the grammar and N^+ stands for the set of all non-empty, finite, sequences of elements from N.

The implemented parsing algorithm is then defined as follows:

1: **for** $j = 1$ to n **do**
2: **for** $i = 1$ to $n - j + 1$ **do**
3: $N1_{i,j} = \{X : X \in N, (X \rightarrow w_{ij}) \in R\}$
4: $N2_{i,j} = \{X\bullet : X \in N, \exists\beta \in N^+, \exists(Z \rightarrow X\beta) \in R, \text{ s.t. } X \rightarrow w_{ij}\}$
5: **for** $j = 2$ to n **do**
6: **for** $i = 1$ to $n - j + 1$ **do**
7: **for** $k = 1$ to $j - 1$ **do**
8: $N1_{i,j} = N1_{i,j} \bigcup$
$\{X : X \in N, \exists(\alpha\bullet \in N2_{i,k}), \exists(Y \in N1_{i+k,j-k}) \text{ s.t. } (X \rightarrow \alpha Y) \in R\}$
9: $N2_{i,j} = N2_{i,j} \bigcup$
$\{\alpha X\bullet : \exists(\alpha\bullet \in N2_{i,k}), \exists(X \in N1_{i+k,j-k}), \exists(\beta \in N^+) \text{ s.t.}$
$\exists(Z \rightarrow \alpha X\beta) \in R)\} \bigcup$
$\{X\bullet : X \in N1_{i,j}, \exists\beta \in N^+ \text{ s.t. } (Z \rightarrow X\beta) \in R\}$

The lines 1-4 correspond to the initialization step and the lines 5-9 correspond to the subsequent filling-up of the chart. When processing word lattices the initialization step is adapted in the same way as discussed in [11, 12].

3 The FPGA Design

Our new design is similar as interface to the external world with our previous designs [11, 12]. Basically, around the FPGA we have a memory that stores the chart and several memories storing identical copies of the CFG for processor lookup. Around each grammar memory we can cluster several processors in order to ensure optimal grammar memory utilisation. Indeed the utilisation of the grammar memories depends on the grammar characteristics and for this reason the possibility to configure the number of processors is an important feature of our design. The number of processors can be adjusted by "profiling" the grammar.

The main differences with the previous designs are (1) we can deal with nplCFG (see section 2) not only CNF CFGs and (2) a more efficient task distribution on the processors.

As in the previous designs, the low average processor load and I/O bandwidth exploitation were a direct consequence of the data-dependencies imposed by the CYK algorithm, we solved this problem by splitting big chunks of data to be processed in smaller parts, called tiles. In the new design, the tiles are distributed over the processors leading to a much more efficient load distribution. Thus, almost all processors are kept busy almost all the time and I/O bandwidth exploitation is also improved due to the continuous reading and writing of the processing results.

4 Design Performance

The performance measurements presented in this section were based on a grammar extracted from the SUSANNE corpus. The FPGA used for synthesis is a

Xilinx Virtex 2000efg1156 that allowed us to connect 16 grammar memories (databus of 16 bits) for grammar lookup. A number of 16 processors were used for this benchmark (one processor per grammar memory).

In order to determine the maximal clock frequency at which the system works, the design was synthesized (with LeonardoSpectrum v2000.1a2) and Placed&Routed (with Design Manager (Xilinx Alliance) Series 2.1i) in the FPGA. The resulting system uses about 60% of the FPGA and was benchmarked (with ModelSim SE-EE 5.4c) at a clock frequency of 60 MHz.

The software used for comparison is a `C` implementation of the same parsing algorithm run on a SUN station (ultra-sparc 60) with 1 processor at 360 MHz. For comparison, 2,022 sentences were parsed and validated (the hardware output

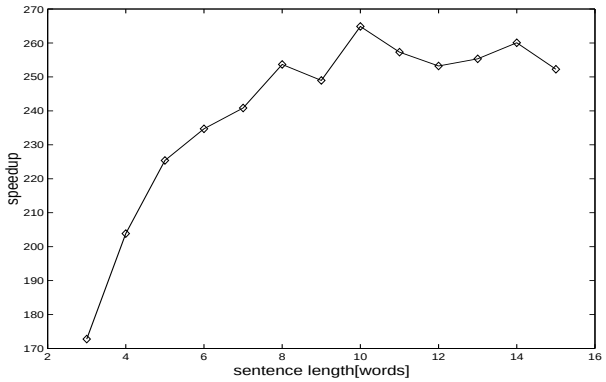

Fig. 1. Hardware speedup. Average over at least 100 sentences for each sentence length.

was compared against the software output to detect mismatches). The sentences had a length ranging from 3 to 15 words and were all taken from the SUSANNE corpus. The average speedup factor was of 244 and figure 1 shows its evolution as a function of the sentence length.

As the produced parse forest was dynamically made available at the output of the hardware, it was also important to determine the data-rate at which an external interface should collect the data in order not to create a bottleneck in the system at this point. The tests showed that the required transmission rate was less than 132 Mbyte/s, which can be achieved with a PCI interface.

5 Conclusions

A design implementing an enhanced version of the CYK algorithm for the parsing of word lattices with almost unrestricted CFG has been presented.

The main conclusions of this work are: (1) in the CYK framework, data-dependencies are the main reason for low processor load; (2) task distribution, as implemented through the tiling mechanism in our design, strongly contributes

to the improvement of the overall processors load; (3) when tiling is used, I/O bandwidth and processor load are directly dependent.

In its current version, our design allows an promising speed-up factor of 244, when compared with a pure software implementation. Additional investigations will be carried out in order to determine the critical parameters that should be further optimized to increase the efficiency and integrability of our hardware parser.

References

1. G. Sampson (1994) The Susanne Corpus Release 3. School of Cognitive & Computing Sciences, University of Sussex, Falmer, Brighton, England.
2. K.H. Chu and K.S. Fu (1982) VLSI architectures for high speed recognition of context-free languages and finite-state languages. Proc. 9th Annu. Int. Symp. Comput. Arch., 43–49.
3. Y. T. Chiang and K.S. Fu (1984) Parallel Parsing Algorithms and VLSI Implementations for Syntactic Pattern Recognition. IEEE Transactions on PAMI.
4. A. V. Aho and J. D. Ullman (1972) The Theory of Parsing, Translation and Compiling. Prentice-Hall.
5. R. Feldman, et al. (1998) Text Mining at the Term Level. Proc. of the 2nd European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD'98). Nantes, France
6. Feldman,R. et al. (1996). Efficient Algorithm for Mining and Manipulating Associations in Texts. *13th* European Meeting on Cybernetics and Research.
7. Schaüble,P. (1997) Mutlimedia Information Retrieval - Content-Based Information Retrieval from Large Text and Audio Databases. Kluwer Academic Publishers.
8. Hull,D., et al. (1996) Xerox TREC-5 Sire Report: Routing, Filtering, NLP and Spanish Tracks. NIST Special Publication 500-238: The Fifth Text REtrieval Conference (TREC-5). Gaithersburg, Maryland.
9. Fu K.S. (1974) Syntactic methods in pattern recognition. Academic Press.
10. J.-C. Chappelier and M. Rajman (1998). A Generalized CYK Algorithm for Parsing Stochastic CFG. TAPD'98 Workshop, 133–137. Paris, France.
11. C. Ciressan, et al. (2000). An FPGA-based coprocessor for the parsing of context-free grammars. 2000 IEEE Symp. on FCCM, Computer Society Press., 236-245.
12. C. Ciressan, et al. (2000). Towards NLP co-processing : An FPGA implementation of a context-free parser. TALN 2000, 99-100, Lausanne, Switzerland.
13. J.-C. Chappelier, M. Rajman (1998). A generalized CYK algorithm for parsing stochastic CFG (TR 98/284). DI-LIA, EPFL