JAIST Repository
https://dspace.jaist.ac.jp/

Title	FPGAを用いたCKYパージングの高速化
Author(s)	伊藤，靖朗
Citation	
Issue Date	2003-03
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/1710
Rights	
Description	Supervisor:中野 浩嗣，情報科学研究科，修士

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

修 士 論 文

FPGAを用いたCKYパージングの高速化

北陸先端科学技術大学院大学
情報科学研究科情報処理学専攻

伊藤 靖朗

2003年3月

修 士 論 文

FPGAを用いたCKYパージングの高速化

指導教官　中野浩嗣 助教授

審査委員主査　中野浩嗣 助教授
審査委員　浅野哲夫 教授
審査委員　金子峰雄 教授

北陸先端科学技術大学院大学
情報科学研究科情報処理学専攻

110015 伊藤 靖朗

提出年月: 2003年2月

Copyright © 2003 by Yasuaki Ito

概 要

本論文では, FPGA を用いた文脈自由文法に対する CKY パージングを高速に実行するハードウェアの実装法を提案する.

FPGA (Field Programmable Gate Array) とは, ユーザによって設計されたハードウェア設計を即座に埋め込むことができるプログラム可能な VLSI である. ユーザのハードウェア論理設計は, FPGA ベンダの設計ツールを用いることによって FPGA に埋め込むことが可能である. とりわけ, 既存のソフトウェアアプローチより高速で効果的な FPGA ベースの手法を開発することが目標である.

CKY パージングとは, 文脈自由文法 G と文字列 x が与えられたときに G が x を導出するかどうか判定するものである. この CKY パージングは, x の長さが n のときに, $O(n^3)$ 時間で導出するかを判定するできることが知られている. 任意の文脈自由文法 G が与えられたときに, その文法に対する CKY パージングを行うハードウェアの Velilog HDL 記述を生成するハードウェアジェネレータを示す. 生成された記述は, FPGA に実装され, 任意の文字列 x に対して, G が x を導出するかを判定する. Xilinx 社の FPGA を用いて実際に動作させ, 性能評価を行った. 結果として, ソフトウェアによる CKY パージングより最大で約 3,000 倍の高速化に成功した.

目 次

第 1 章 はじめに **1**
1.1 背景 1
1.2 FPGA 1
1.3 部分計算 2
1.4 CKY パージングアルゴリズム 2
1.5 全体の流れ 2
1.6 本論文の構成 3

第 2 章 CKY パージング **4**
2.1 文脈自由文法 4
2.2 チョムスキー標準形 4
2.3 パージング問題 4
2.4 CKY パージング 5

第 3 章 CKY パージングの評価 **7**
3.1 ソフトウェアアルゴリズム 7
3.1.1 Naive アルゴリズム 7
3.1.2 Table アルゴリズム 7
3.2 ハードウェアアルゴリズム 8
3.3 性能評価 10
3.4 まとめ 12

第 4 章 プロトタイプの作成 **13**
4.1 CKY パージング回路 13
4.2 回路の並列化 14
4.3 性能評価 16
4.4 まとめ 18

第 5 章 実用的な CKY パージングの実装 **21**
5.1 ソフトウェアアプローチ 21
5.2 ハードウェアアプローチ 21

5.2.1 ハードウェア実装 1 21
5.2.2 ハードウェア実装 2 21
5.3 性能評価 23
5.4 まとめ 25

第 6 章 おわりに 26
6.1 本研究の成果 26
6.2 今後の課題 26

謝辞 27

参考文献 28

論文リスト 30

Appendix 31

第1章　はじめに

1.1　背景

文脈自由文法によって記述された形式言語は, パターン認識やプログラミング言語及び自然言語処理などの多くのアプリケーションで使用される. そのような形式言語に対する解析速度は, アプリケーションの実装において重要な問題となる. 例えば, 自然言語処理アプリケーションの特別なケースでは, 実時間の制約を考慮する必要があり, 効率的な解析法を提案する必要がある. そのようなアプリケーションの典型的な例を以下に示す.

- データ処理： 情報検索や文章抽出を行う際に, 光学文字認識 (Optical Character Recognition) をすると同時にスペルチェックを行うような技術では, 言語に関する構文情報の統合により, 構文解析の一層の処理速度向上を要求する可能性がある. このような巨大な量のデータを処理する必要がある場合, 効率的かつ単純構造の解析プロセッサが要求される.
- ヒューマンマシン・インターフェース：音声認識インターフェースでは, リアルタイムに解析を行う必要があり, 構文解析の性能向上が必須である.

1.2　FPGA

FPGA (Field Programmable Gate Array) とは, ユーザによって設計されたハードウェア設計を即座に埋め込むことができるプログラム可能な VLSI である. 典型的な FPGA は, 書き換え可能なロジックセルの配列, 分散したメモリブロック, そしてそれらを結合するプログラム可能な配線から成る. ロジックセルは, 通常 2 入力論理関数もしくは 4 入力 1 出力のマルチプレクサ, 幾つかのフリップフロップを持つ. メモリブロックは, 別々のアドレスに対して同時にデータの読み込みと書き込みが可能なデュアルポート RAM である. ユーザのハードウェア論理設計は, FPGA ベンダの設計ツールを用いることによって FPGA に埋め込むことが可能である. 本研究では, 有用な計算を高速化するために FPGA を使用する. とりわけ, 既存のソフトウェアアプローチより高速で効果的な FPGA ベースの手法を開発することが目標である.

1.3 部分計算

本研究では, **部分計算**[10] の概念に基づき, FPGA を用いた計算の高速化を行う. 与えられた問題を解くために評価する関数を $f(y,x)$ とする. ただし, その関数は y を固定して繰り返し評価されることが多いとする. その場合, $f_y(x)=f(y,x)$ のようなインスタンスに特化した関数 f_y を評価することによって, $f(y,x)$ の計算の簡単化が可能である. 本研究のアイディアは, 固定した y と変数 x に対して $f_y(x)$ を計算するために最適化したハードウェアを作成することである. つまり, 次の 2 つの性質を満たす $f(y,x)$ が必要となる問題に対し, 問題のインスタンスに特化した手法を FPGA を用いて示していく.

1. 固定した y の値が問題のインスタンスに依存する

2. 問題を解くために様々な値をとる変数 x に対して $f(y,x)$ の値が繰り返し評価される

1.4 CKY パージングアルゴリズム

本論文では, 先に示した FPGA ベースのアプローチを用いて, 文脈自由文法のパージング [12] を文法を固定することにより高速化するハードウェアを示す. 入力文字列 x の長さを n としたとき, *CKY(Cocke-Kasami-Younger)* **パージング**は, $O(n^3)$ 時間で計算することはよく知られている [1]. 文脈自由言語のパージングは, 自然言語処理 [5, 15], コンパイラ [1], バイオインフォマティックス [14] など様々な分野において多くのアプリケーションが存在する. 幾つかの研究において, 文脈自由言語のパージングの高速化が行われてきた [4, 9, 11, 15]. 長さ n の文字列に対するパージングが PRAM 上で n^6 台のプロセッサを用いて $O((\log n)^2)$ 時間で行われることが示された [9]. また, メッシュ結合のプロセッサ列を用いることによって, パージングが n 台のプロセッサを用いて $O(n^2)$ 時間, n^2 台のプロセッサを用いて $O(n)$ 時間で実行可能である [11]. これらの並列アルゴリズムは少なくとも n 台のプロセッサが必要なので大きな n に対しては非現実的である. Ciressan ら [6, 7] は, 文脈自由文法の制限したクラスに対して CKY パージングを行うハードウェアを示し, FPGA を用いてテストを行った. しかしながら, ハードウェア設計と制御アルゴリズムはメッシュ結合したプロセッサ上で行うもの [11] と本質的に同じであり, インスタンスに特化してはいなかった.

1.5 全体の流れ

文脈自由文法のパージングを行うインスタンスに特化した手法のために, 任意の文脈自由文法 G に対する CKY パージングを行う Verilog HDL ソースを生成するハードウェアジェネレータを提案する. G を文脈自由文法, x を文字列, $f(G,x)$ をブール変数を返す関数とする. ただし $f(G,x)$ は, G が x を導出しかつそのときに限り TRUE を返すものとする. 生成された Verilog HDL ソースは, Xilinx 社の ISE ロジックデザインツール [17] を用いて

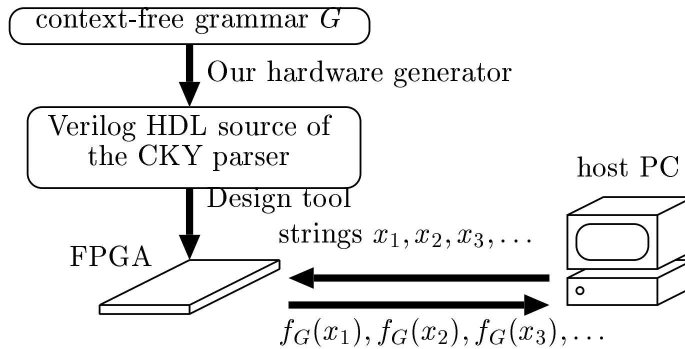

図 1.1: Our hardware parsing system.

コンパイルを行う. そして得られたオブジェクトファイルを Xilinx Virtex-II シリーズの FPGA[16] にダウンロードすると, FPGA は文法 G を固定した $f_G(x)$ を計算する回路となる. つまり入力文字列 x に対して G が x を導出するかどうかを調べる回路になる. 図 1.1 に, CKY パージングのシステムを示す. ホスト PC によって文字列 $x_1, x_2, x_3, \ldots$ が与えられ, FPGA はこれらの文字列が G で導出可能かどうか, すなわち $f_G(x_1), f_G(x_2), f_G(x_3), \ldots$ の結果を返す.

どれだけ FPGA が CKY パージングを高速化しているかを明らかにするために, ソフトウェアと比較する. このソフトウェアのアルゴリズムは, p 個の生成規則全てを $O(p)$ 時間で調べることによって計算を行う. このアルゴリズムを用いた CKY パージングは $O(n^3p)$ 時間で計算可能である. 一方, FPGA を用いた手法は CKY パージングを $O(n^3 \log b)$ 時間で計算可能である. ただし文脈自由文法の非終端期号数を b とする. 常に $b \leq p$ より, 理論的には本研究の提案手法はソフトウェアアプローチよりも高速である.

そこで, 実際に Vertex-II シリーズの FPGA を用いて性能評価をし, ソフトウェアと比較した. その結果, ソフトウェアに対して提案手法は最大で約 3,000 倍の高速化に成功した.

1.6 本論文の構成

本論文は, 次のように構成される. 第 2 章では, CKY パージングについて述べる. 第 3 章では, CKY パージングの評価について説明する. 第 4 章では, CKY パージングを行うハードウェア実装を紹介し, FPGA を用いて実際に動作を確認した. 第 5 章では, 実用的なサイズの CKY パージングのハードウェア実装を示し, 実際に FPGA を用いて動作させた. 第 6 章では, 本研究の成果と今後の課題について述べる.

第2章　CKYパージング

本章では, CKY パージングについて説明をする.

2.1　文脈自由文法

文脈自由文法(Context-Free Grammer) とは 4 つ組

$$G = \{N, \Sigma, P, S\}$$

によって定義される文法で, 各成分は次のものとする.

N: 非終端記号の集合

Σ: 終端記号の集合

P: 生成規則の集合

S: 開始記号. ただし $S \in N$

2.2　チョムスキー標準形

P にある各生成規則が

$$\begin{aligned} A &\rightarrow BC\ (A, B, C \in N) \\ A &\rightarrow a\ (A \in N, a \in \Sigma) \end{aligned}$$

のどちらかの形式であるような文脈自由文法を, **チョムスキー標準形** (Chomsky Normal Form (CNF)) と呼ぶ.

2.3　パージング問題

今回扱ったパージング問題とは, 与えられたチョムスキー標準形の文脈自由文法 G と Σ 上の入力文字列 x に対して, 開始記号 S が x を導出するかどうか決定する問題である. 例えば, $G_{\text{example}} = (N, \Sigma, P, S)$ が次のような文法であるとする.

- $N = \{A, B, C\}$
- $\Sigma = \{a, b\}$
- $P = \{S \to AB, S \to BA, S \to SS, A \to AB, B \to BA, A \to a, B \to b\}$

このとき S が次のように $abaab$ を導出するので, 文脈自由文法 G は $abaab$ を導出する.

$$S \Rightarrow AB \Rightarrow ABA \Rightarrow ABAA \Rightarrow ABAAB \Rightarrow \cdots \Rightarrow abaab$$

2.4 CKY パージング

チョムスキー標準形の文脈自由文法 G と文字列 x に対して G が x を導出するかどうか決定する CKY パージング法について説明する. 文字列 x を長さ n の文字列 $x = x_1x_2\cdots x_n$ とする. ただし, 各 $x_i (1 \le i \le n)$ は Σ の要素である. また, N の部分集合を $T[i,j] (1 \le i \le j \le n)$ で表し, $T[i,j]$ の各要素は部分文字列 $x_ix_{i+1}\cdots x_j$ を導出するものとする. CKY パージングのアイディアは, 次の関係を用いて, 各 $T[i,j]$ を計算することである.

$$\begin{aligned} T[i,i] &= \{A \mid (A \to x_i) \in P\} \\ T[i,j] &= \bigcup_{k=i}^{j-1} \{A \mid (A \to BC) \in P, B \in T[i,k], \text{ and } \\ & \quad C \in T[k+1,j]\} \end{aligned}$$

2 次元配列 T は, *CKY* **テーブル** と呼ばれる. 文法 G が文字列 x を生成しかつそのときに限り, $T[1,n]$ に S が含まれる. N の部分集合 U, V 間のバイナリ演算 $\otimes_G$ を $U \otimes_G V = \{A \mid (A \to BC) \in P, B \in U, \text{and } C \in V\}$ と定義する. この $\otimes_G$ を用いて CKY パージングの詳細を次に示す.

CKY **パージング**

1. $N[i,i] \leftarrow \{A \mid (A \to x_i) \in P\}$ for every i $(1 \le i \le n)$
2. $N[i,j] \leftarrow \emptyset$ for every i and j $(1 \le i < j \le n)$
3. for $j \leftarrow 2$ to n do
4.   for $i \leftarrow j-1$ downto 1 do
5.   for $k \leftarrow i$ to $j-1$ do
6. $N[i,j] \leftarrow N[i,j] \bigcup (N[i,k] \otimes_G N[k+1,j])$

最初の 2 行は CKY テーブルを初期化し, 次の 4 行は CKY テーブルの計算を行う. 図 2.1 は, G_{example} と文字列 $abaab$ に対する CKY テーブルを図示したものである. $S \in T[1,5]$ より, G_{example} が文字列 $abaab$ を導出することがわかる.

j \ i	1	2	3	4	5
5	S, A	S, B	$\emptyset$	S, A	B
4	S, A	S, B	$\emptyset$	A	b
3	S, A	S, B	A	a	
2	S, A	B	a		
1	A	b			
	a				

図 2.1: The CKY table for G_{example} and $abaab$.

最後の 4 行が CKY パージングの計算の大半を占めることは明らかである. 6 行目を計算するのに必要な時間を t とすると, 3–6 行目の計算時間は,

$$\sum_{j=2}^{n-1}\sum_{i=1}^{j-1}\sum_{k=i}^{j-1} t = t\sum_{j=2}^{n-1}\sum_{i=1}^{j-1}(j-i) = \frac{1}{6}t(n^3 - 3n^2 + 2n)$$

となる.

第3章　CKYパージングの評価

本章では, 生成規則 P の各規則 $A \to BC$ に対して, $B \in U$ かつ $C \in V$ であるかどうか調べる, つまり $U \otimes_G V$ を計算するアルゴリズムに焦点を当てて説明する.

3.1　ソフトウェアアルゴリズム

この節では, 2 章で述べた任意の非終端記号の集合 U と V に対して $U \otimes_G V$ の計算を行う逐次 (ソフトウェア) アルゴリズムを 2 つ示す.

3.1.1　Naive アルゴリズム

最初のアルゴリズムは, 生成規則 P の各規則 $A \to BC$ に対して, $B \in U$ かつ $C \in V$ であるかどうか調べるアルゴリズムである. 適当なデータ構造を用いることによって, $O(1)$ 時間でこれを行うことが可能なことは明らかである. よって, P にある $A \to BC$ 形式の生成規則数を p とすると, $U \otimes_G V$ は $O(p)$ 時間で評価可能である. 以上より, このアルゴリズムを用いると CKY パージングの計算時間は $O(n^3 p)$ 時間である. 今後このアルゴリズムを *Naive* **アルゴリズム**と呼ぶ.

3.1.2　Table アルゴリズム

2 番目のアルゴリズムは $U \otimes_G V$ の計算にルックアップテーブルを用いるアルゴリズムで, 今後 *Table* **アルゴリズム**と呼ぶ. このアルゴリズムは, 各 U, V のすべての組合せに対して $U \otimes_G V$ の値をあらかじめ計算しテーブルに記録しておき, $U \otimes_G V$ を計算するときにはそのテーブルを参照する. 非終端記号の集合 N が b 個の非終端記号を持つとし, $N = \{N_1, N_2, \ldots, N_b\}$ であるとする. 与えられた U $(\in 2^N)$ を b ビットベクトル $u_1 u_2 \cdots u_b$ であらわし, 各 i $(1 \leq i \leq b)$ について $u_i = 1$ でありかつそのときに限り $N_i \in U$ であるとする. 同様に, V $(\in 2^N)$ も b ビットベクトル $v_1 v_2 \cdots v_b$ であらわす. $U \otimes_G V$ を計算するためには, メモリ上に $2^{2b} \times b$ ビットのテーブルが必要である (つまり, アドレスが $2b$ ビットでデータが b ビットのメモリ). そのテーブルの $u_1 u_2 \cdots u_b v_1 v_2 \cdots v_b$ 番目のエントリーは, $w_1 w_2 \cdots w_b$ を記録される. このとき $w_1 w_2 \cdots w_b$ は, $W = U \otimes_G V$ を表す b ビットのベクトルである. そのようなテーブルが利用できれば, $U \otimes_G V$ が $O(1)$ 時間で計算

ができることは明らかである．しかしながら, b がそれほど大きくないときでもそのテーブルは巨大になってしまう．P に $b = 64$ 個の非終端記号があるとき, テーブルのサイズは $2^{2 \cdot 64} \times 64 = 2^{134} \approx 10^{40}$ ビットになり, 極めて巨大になる.

そこでテーブルのサイズを減らすために Table アルゴリズムを変更する．N を $N^i = \{N_{c(i-1)+1}, N_{c(i-1)+2}, \ldots, N_{ci}\}, (1 \leq i \leq \frac{b}{c})$ であらわされる同じサイズの部分集合に分割する．つまり, 集合 N を各部分集合が c 個の非終端記号を含むように $\frac{b}{c}$ 個の部分集合に分割する．c は 0 より大きな整数値をとるが, 実際には 16 より大きくなることはない．$U \otimes_G V$ を求めるために次の $(\frac{b}{c})^2$ 個のバイナリ操作 $\otimes_G^{i,j}$ $(1 \leq i, j \leq \frac{b}{c})$ を用いる.

- $\otimes_G^{i,j}$ is $2^{N^i} \times 2^{N^j} \rightarrow 2^N$, and
- $(U \cap N^i) \otimes_G^{i,j} (V \cap N^j) = \{A \mid (A \rightarrow BC) \in P, B \in U \cap N^i, \text{ and } C \in V \cap N^j\}$.

よって次のように表すことができる.

$$U \otimes_G V = \bigcup_{1 \leq i,j \leq (\frac{b}{c})^2} (U \cap N^i) \otimes_G^{i,j} (V \cap N^j).$$

このように i と j のすべての組合せに対して $\otimes_G^{i,j}$ を評価することによって, $\otimes_G$ を計算できる．前述のとおり, $\otimes_G^{i,j}$ は, サイズ $2^{2c} \times b$ のテーブルを参照することによって計算可能である．よって $\otimes_G$ は, $(\frac{b}{c})^2$ 個のテーブルを参照することによって $O((\frac{b}{c})^2)$ 時間で求めることが可能である．テーブル全体のサイズは, $\frac{b^3}{c^2}2^{2c}$ ビットである．$b = 64, c = 8$ のとき, テーブルは $2^{28} = 256$ M ビットとなり実現可能である．しかしながら, $(\frac{b}{c})^2 = 64$ 回テーブルを参照する必要がある．テーブルのサイズとその参照回数は, 生成規則の数 p に依存しないことに注意する．このように, Table アルゴリズムは p の値が大きい場合でも効果的である.

3.2 ハードウェアアルゴリズム

FPGA を用いて $\otimes_G$ を計算する回路を構築することによって $\otimes_G$ の評価を高速化することについて説明する．$\otimes_G$ を計算する回路を今後 *CKY* **回路**と呼ぶことにする．各 U, V $(\in 2^N)$ をそれぞれ b ビットのビット列 $u_1 u_2 \cdots u_b$ と $v_1 v_2 \cdots v_b$ で表すことにする. CKY 回路は $W = U \otimes_G V$ を求める回路, つまり U と V から $w_1 w_2 \cdots w_b$ を計算する回路とする．次に w_k を求める方法を示す．$N_k \rightarrow N_{i_1} N_{j_1}, N_k \rightarrow N_{i_2} N_{j_2}, \ldots, N_k \rightarrow N_{i_s} N_{j_s}$ が生成規則 P の中で非終端記号 N_k を左側にもつ生成規則であるとする．w_k は次式,

$$w_k = (u_{i_1} \wedge v_{j_1}) \vee (u_{i_2} \wedge v_{j_2}) \vee \cdots \vee (u_{i_s} \wedge v_{j_s})$$

で計算される.

そこで, 生成規則から上記の式に対応するハードウェア記述言語を生成するプログラムの作成を行った．このプログラムは, テキストファイルに書かれた生成規則から各 w_k を

```
 1 module comp(u,v,w);
 2   input [3:1] u,v;
 3   output [3:1] w;
 4
 5   assign w[1] = (u[2] & v[3])
 6               | (u[3] & v[2])
 7               | (u[1] & v[1]);
 8   assign w[2] = (u[2] & v[3]);
 9   assign w[3] = (u[3] & v[2]);
10 endmodule
```

図 3.1: Verilog HDL によるサブモジュールの記述例

計算する回路記述を出力する．今回は, C や Pascal のようなハードウェア記述言語である Verilog-HDL を用いて回路記述を行った．この回路は, メモリアクセスの制御部や PC と FPGA 間のインターフェイスを含むメインモジュールに対するサブモジュールに相当する．生成されたサブモジュールの記述例を図 3.1 に示す．

1 行目は, モジュールの名前とそのモジュールの入力と出力と名前の定義である．その入力と出力の詳細が 2 行目と 3 行目に定義されている．5 行目から 9 行目では, 生成規則に従って出力ベクトルの各エントリーの計算を行う．このサブモジュールの回路図を図 3.2 に示す．先に示したとおり, 2 入力の s 個の AND ゲートと $s-1$ 個の OR ゲートで構成される組合せ回路によって, w_k を計算することが可能である．この回路の深さ (組合せ回路の入力から出力までのゲート数) は, $\lceil \log(s-1) \rceil + 1$ である．生成規則 P の中で $A \to BC$ の形式が p 個あるとき, p 個の AND ゲートと $p-b$ 個の OR ゲートで構成される組合せ回路で $w_1 w_2 \cdots w_b$ を計算することが可能である．常に $s \leq b^2$ だから, 回路の深さは $\lceil \log(b^2-1) \rceil + 1 \leq 2\log b + 1$ より深くなることはない．以上より, CKY パージングは, この回路を用いることによって $O(n^3 \log b)$ 時間で行うことができる．図 3.2 は, $\otimes_{G_{\mathrm{example}}}$ の回路を図示したものである．G_{example} は 5 個の生成規則と 3 個の非終端記号があるから, 回路は 5 個の AND ゲートと $5-3=2$ 個の OR ゲートから構成される．

3.1.1 節で示した Naive アルゴリズムは $\otimes_G$ を計算するのに $O(p)$ 時間, 3.1.2 節で示した Table アルゴリズムは $O((\frac{b}{c})^2)$ 時間必要であることを示した．一方, $\otimes_G$ に対する回路では $O(\log b)$ に比例した遅延時間である．常に $b \leq p \leq b^3$ が成立するので, $\otimes_G$ に対する回路は理論上逐次アルゴリズムよりも高速である．

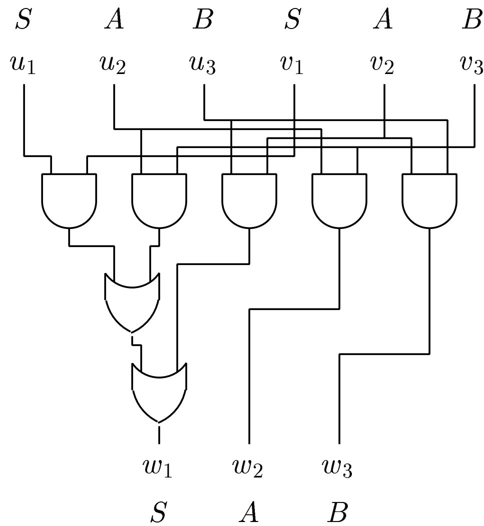

図 3.2: The circuit for computing $\otimes_{G_{\mathrm{example}}}$.

3.3 性能評価

$\otimes_G$ を計算するハードウェアとソフトウェアを実装し, 性能評価を行った. このとき, タイミング解析には Altera 社 [2] の Quartus II[3], 回路のテストには Altera 社の APEX20K シリーズの FPGA(200KB の内部メモリと 16K のロジックエレメントを持つ 40 万ゲート相当の EP20K400EBC652-1X) を用いた. 性能を評価するために, 次に示す環境で実行し性能を計測した.

CPU: Intel Xeon 1.7GHz

メモリ: RDRAM 2GB

OS: Linux Kernel 2.4.9

関数 $\otimes_G$ を計算するハードウェアとソフトウェアの実行時間のグラフを図 3.3 に示す. 図より, 64 ビットのベクトルを用いた実装より 32 ビットのベクトルを用いた実装の方が実行時間が短いことがわかる. 実験に用いた PC は 32 ビット CPU なので, 1 ワードで 32 ビットのベクトルを表現することが可能である. 今回の実装では, 非終端記号の集合をビットのベクトルであらわし, 非終端記号の数が 32 のときは 32 ビットのベクトル, 非終端記号の数が 64 のときは 64 ビットのベクトルを用いた. このため 64 ビットの実装では 2 ワードで計算を行う必要があるので, 32 ビットの実装と比べて余分なオーバーヘッドが存在する. よって 64 ビットのベクトルを用いた実装より 32 ビットのベクトルを用いた実装の方が実行時間が短い.

Naive アルゴリズムはすべての生成規則 $A \to BC$ について $B \in U$ かつ $C \in V$ であるかどうか調べるアルゴリズムなので, Naive アルゴリズムの計算時間は生成規則数に比例

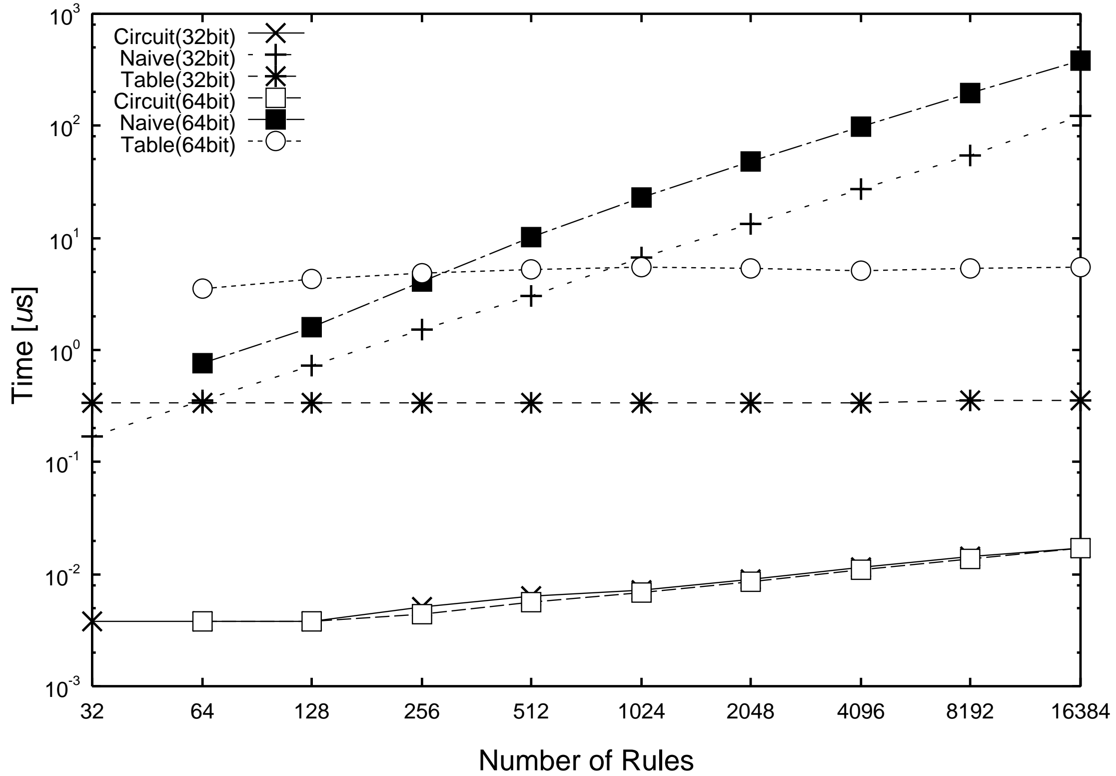

図 3.3: Computing time to evaluate $\otimes_G$.

している．

一方, Table アルゴリズムの実行時間は生成規則数に依存せず, テーブルのアクセス回数に依存する．よって Table アルゴリズムの実行時間は生成規則数が変わってもだいたい一定である．また Table アルゴリズムは $\otimes_G$ を求めるために, 非終端記号数が b のとき $(\frac{b}{c})^2$ 回テーブルを参照する必要がある．このため b が大きくなるに従って実行時間も長くなっている．生成規則数 p が少ないとき Naive アルゴリズムの方が Table アルゴリズムより実行時間が短いが, p の数が多くなるにつれて Table アルゴリズムの方が Naive アルゴリズムよりかなり高速に実行される．

$\otimes_G$ を計算するのに, 本研究で用いたハードウェアの手法では 64 ビットのベクトルを用いた場合 Table アルゴリズムと比べほぼ 1,000 倍の高速化を達成した．また, 32 ビットのベクトルを用いた場合は 100 倍近くの高速化が得られた．Naive アルゴリズムと比較すると $p = 16{,}384$ のとき最も差があらわれ, 64 ビットのベクトルのとき約 22,000 倍, 32 ビットベクトルのとき約 7,300 倍ハードウェアが Naive アルゴリズムより高速である．本研究で用いたハードウェアの実装では, 非終端記号数をあらわすビットのベクトルのサイズに実行時間は依存しない．よって, 64 ビットのベクトルを用いたときと 32 ビットのベクトルを用いたときのハードウェアの実行時間はほとんど同じである．

3.4 まとめ

3.3 節より, 本研究で提案したハードウェア手法がソフトウェア手法と比べて有効であることがわかった. このハードウェア手法を用いた CKY パージングの実装を次章で説明する.

第4章　プロトタイプの作成

本章では, 前章で説明した CKY 回路を用いて CKY パージングを行うハードウェア実装の説明を行う.

4.1　CKY パージング回路

ここでは, CKY パージングを計算する回路の説明を行う. この回路の基本的な構成要素を次に示す.

- b ビット n^2 ワード (デュアルポート) メモリ
- b ビット n ワード (デュアルポート) メモリ
- $\otimes_G$ を計算する CKY 回路
- b 個の OR ゲート列
- b ビットレジスタ

図 4.1 に CKY パージング回路のブロック図を示す. b ビット n^2 ワードメモリは, CKY テーブルの内容が記憶される. 入力 $T[1,1]$, $T[2,2]$, $\ldots, T[n,n]$ が b ビット n^2 ワードメモリに入力される. b ビット n ワードメモリには, 処理をしている CKY テーブルの 1 行分が記憶される. つまり, CKY テーブルの j 行目 $T[1,j], T[2,j], \ldots$ が記憶される. このとき j は, CKY パージングの 3 行目に現れる変数 j である. b ビットレジスタは, CKY パージングの 6 行目で計算される $T[i,j]$ が格納される. b 個の OR ゲート列は, 6 行目の "$\bigcup$" を計算するために用いられる. b ビット n^2 ワードメモリは CKY 回路に $T[i,k]$ を表している b ビットのベクトルを入力として与える. 同様に, b ビット n ワードメモリは, $T[k+1,j]$ に対する b ビットのベクトルを出力する. CKY 回路は, それらを受け取り, $T[i,k] \otimes_G T[k+1,j]$ に対する b ビットベクトルを計算する. このハードウェア実装を用いることで, CKY パージングの 6 行目は 1 クロックサイクルで計算される. これより, CKY パージングは n^3 クロックサイクルで行うことが可能である. さらに, 実装において 1 クロックサイクルは $O(\log b)$ に比例する. よって, 計算時間は $O(n^3 \log b)$ である.

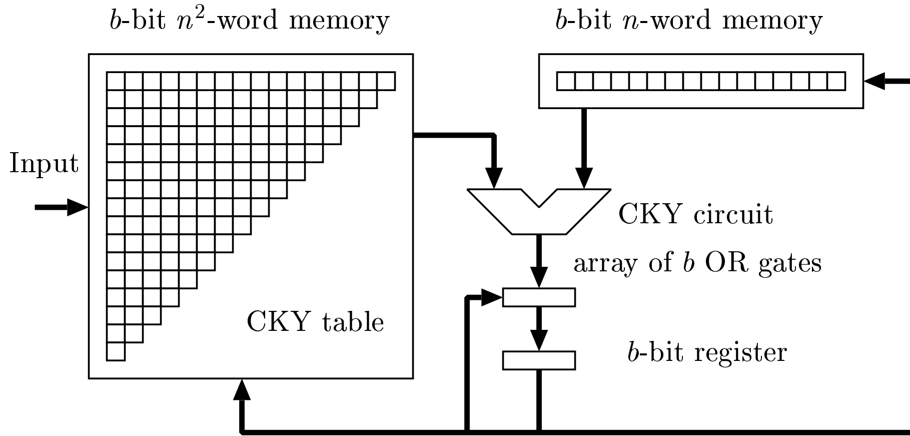

図 4.1: A hardware implementation for the CKY parsing.

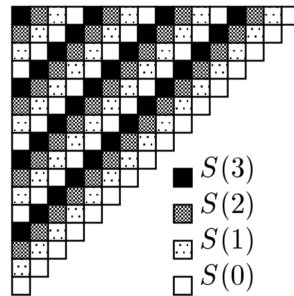

図 4.2: Partitioning the CKY table.

4.2 回路の並列化

ここでは, 複数の CKY 回路を用いた CKY パージングの並列化について説明する. このとき, 並列に CKY テーブルにアクセスする必要があるので, CKY テーブルを m 個のサブテーブル $S(0), S(1), \ldots, S(m-1)$ に分割する. このとき, $S(l)$ は, $(j-i) \bmod m = l$ を満たすように $T[i,j]$ を格納する. 図 4.2 に CKY テーブルを 4 分割したときの図を示す. 図を見るとわかるように, CKY テーブルの各列にある連続した m 個の要素 $T[i,k], T[i,k+1], \ldots, T[i,k+m-1]$ は, 別々のサブテーブルに格納されている. このとき各サブテーブルが異なるメモリバンクに格納されていれば, 連続した m 個の要素に同時にアクセスすることが可能である. この CKY テーブルの分割により, m 個の CKY 回路を用いて CKY パージングの並列化を可能にする. 上記の並列化の性能を評価するために, 複数の CKY 回路を用いて CKY パージングを行う回路の実装を行った.

並列化した CKY パージング回路で用いた基本的な構成要素を次に示す.

– b ビット $\frac{n^2}{m}$ ワード メモリバンク (デュアルポート) m 個

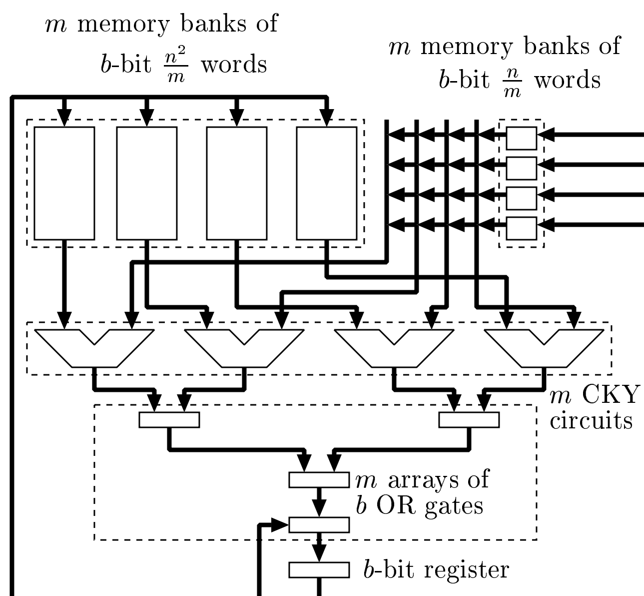

図 4.3: Parallel implementation of the CKY parsing for $m = 4$.

– b ビット $\frac{n}{m}$ ワード メモリバンク (デュアルポート) m 個

– $\otimes_G$ を計算する CKY 回路 m 個

– b 個の OR ゲート列 m 個

– b ビットレジスタ

図 4.3 に並列化した CKY パージング回路の実装を示す．m 個の b ビット $\frac{n^2}{m}$ ワード メモリバンクには m 個のサブテーブルが格納され, 各サブテーブルにつき 1 つのバンクが用いられる．また, m 個の b ビット $\frac{n}{m}$ ワード メモリバンクには, 処理をしている CKY テーブルの 1 行分が記憶される．つまり, $T[i,j]$ の計算を行っているとき CKY テーブルの j 行目 $T[1,j], T[2,j], \ldots, T[j,j]$ が記憶される．このとき, l 番目のバンク $(0 \leq l \leq m)$ に $T[l+1,j]$,$T[l+m+1,j]$,$T[l+2m+1,j], \ldots$ が格納される．$T[1,1]$, $T[2,j]$, $T[1,2]$, $T[3,j]$, $\ldots$, $T[1,m]$, $T[m+1,j]$ が異なるメモリバンクに格納されているから, $\otimes_G$ の m 個の 評価 $T[1,1] \otimes_G T[2,j]$, $T[1,2] \otimes_G T[3,j]$, $\ldots$, $T[1,m] \otimes_G T[m+1,j]$ が 1 クロックサイクルで評価可能である．以上のことより, CKY パージングを m 倍高速化することが可能になる．よって CKY パージングの計算時間は, $m \leq n$ のとき $O(\frac{n^3 \log b}{m})$ である．

4.3 性能評価

CKY パージングを行うハードウェアのプロトタイプの作成を行い, 性能評価を行った. このとき 3.3 節と同様に, タイミング解析には Quartus II, 回路のテストには APEX20K シリーズの FPGA(200KB の内部メモリと 16K のロジックエレメントを持つ 40 万ゲート相当の EP20K400EBC652-1X) を用いた. 性能を評価するために, 次に示す環境で実行し性能を計測した.

CPU: Intel Xeon 1.7GHz

メモリ: RDRAM 2GB

OS: Linux Kernel 2.4.9

非終端記号 b と入力文字列長 n が $b = 32, n = 32$ のときの CKY パージングの計算時間を図 4.4 に示す. ハードウェアは CKY 回路を 1 個もつ回路 (Single-circuit), 2 個もつ回路 (Double-circuit), 4 個もつ回路 (Quad-circuit) を用意し, ソフトウェアと実行時間の比較を行った. ソフトウェアは, Naive と Table アルゴリズム共に図 3.3 に見られるパターンと同じである. これは, $\otimes_G$ で費やされる時間が計算時間の大部分を占めるからである. Single-circuit も図 3.3 に見られるパターンとだいたい同じである. また 4.2 節で示したとおり, Double-circuit もしくは Quad-circuit は実際に高速化されていることがわかる.

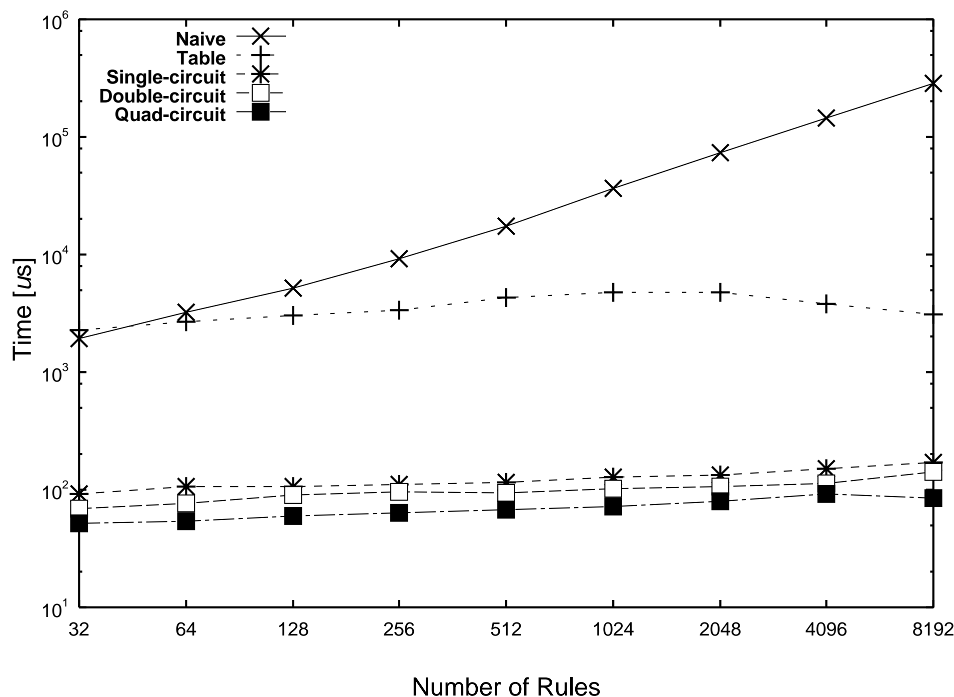

図 4.4: Computing time of the CKY algorithm with $b = 32$ and $n = 32$.

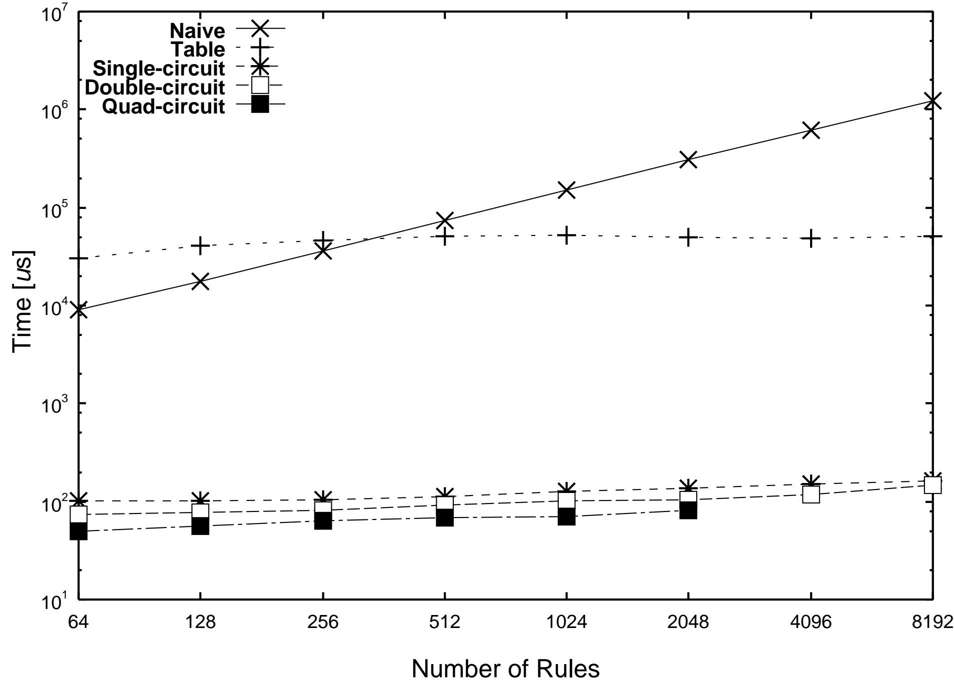

図 4.5: Computing time of the CKY algorithm with $b = 64$ and $l = 32$.

次に, $b = 64, n = 32$ のときのCKYパージングの計算時間を図4.5に示す. $b = 32, n = 32$ のときCKYパージングの実行時間のグラフとだいたい同じパターンであることがわかる. 前に述べたように, ソフトウェアで64ビットベクトルを用いる方法では余分なオーバーヘッドが加わる. しかしハードウェア実装では, この余分なオーバーヘッドは発生しない. その結果, ソフトウェアの実行時間は長くなっていることがわかる. $p = 2{,}048$ の Quad-circuit を構築するのに, 約 9,600 個のロジックブロックが必要である. $p = 4{,}096$ の Quad-circuit を構築するのに必要なロジックブロックは, 今回使用したFPGAの持つロジックブロック数を超えてしまう. よって, Quad-circuit は $p = 2{,}048$ までである.

Table アルゴリズム (ソフトウェア手法) に対するハードウェア手法のスピードアップ率を示した表を表4.1に示す. $b = 32, n = 32$ のとき, 本研究で実装したハードウェア手法は Single-circuit で約 40 倍, Double-circuit で約 50 倍, Quad-circuit で約 70 倍の高速化を達成した. さらに $b = 64, n = 32$ のとき, Single-circuit で約 460 倍, Double-circuit で約 580 倍, Quad–circuit で約 750 倍の高速化を達成した. 以上の結果より, 本研究のCKYパージングを行うハードウェアの手法は実際に有効な手法であることが言える.

$b = 32$ のときの Single-circuit, Double-circuit, Quad-circuit を構成するのに必要なロジックブロックの数を図4.6に示し, $b = 64$ のときのを図4.7に示す. 生成規則が増えるに従ってロジックブロックの数が増加していることがわかる. 先に述べた通り, $b = 64$ のときの Quad-circuit は $p = 4{,}096$ のとき使用したFPGAの持つロジックブロック数を超えてしまうので $p = 2{,}048$ までである.

表 4.1: Speed-up of the CKY hardware approach over the CKY table algorithm.

	$b = 32, n = 32$			$b = 64, n = 32$		
p	Single	Double	Quad	Single	Double	Quad
32	25	33	44	–	–	–
64	25	35	50	304	419	611
128	29	34	51	395	519	731
256	30	35	53	441	577	730
512	37	46	64	454	552	736
1,024	38	48	66	413	513	742
2,048	36	45	60	362	475	600
4,096	26	34	41	326	418	–
8,196	18	22	37	314	348	–

次に, $b = 32$ のときの Single-circuit, Double-circuit, Quad-circuit の最大動作周波数を図 4.8 に示し, $b = 64$ のときのを図 4.9 に示す. 生成規則が増えるに従って最大動作周波数が減少していることがわかる. これは, 生成規則が多くなると回路の深さが深くなり, 回路遅延が増大するからである. このことは, CKY アルゴリズムの計算時間に大きく影響する.

4.4 まとめ

本章では, CKY パージングを高速化に実行するハードウェア手法を提案した. また, 実際に APEX20K シリーズの FPGA を用いて動作を確認し, 性能評価を行った. ハードウェアの性能を評価するために, ソフトウェアの手法を実装し性能を計測した. その結果より, ソフトウェアの手法と比べて最大で約 750 倍の高速化に成功した. しかし, 実際の例として英語の CKY パージングを行うときには入力文字列長は約 100, 非終端記号数は約 200, 生成規則数は約 17,000 で, 本章で扱ったものよりもかなり大規模なものである [13]. よって 5 章では, 実用的なハードウェア実装について述べる.

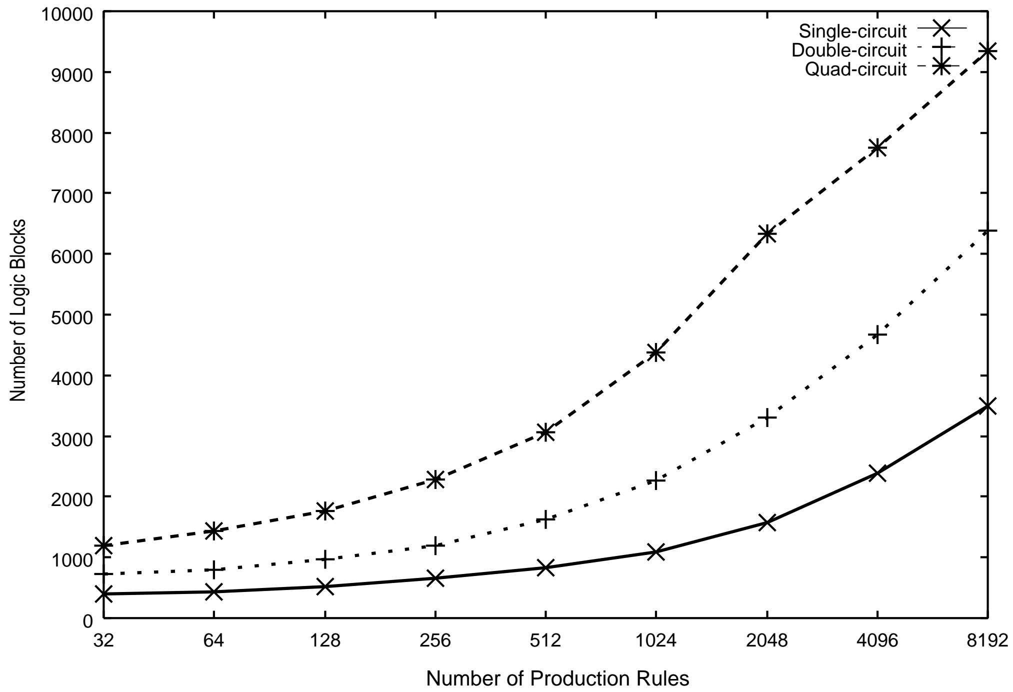

図 4.6: Number of logic blocks used to compute the CKY algorithm with $b = 32$

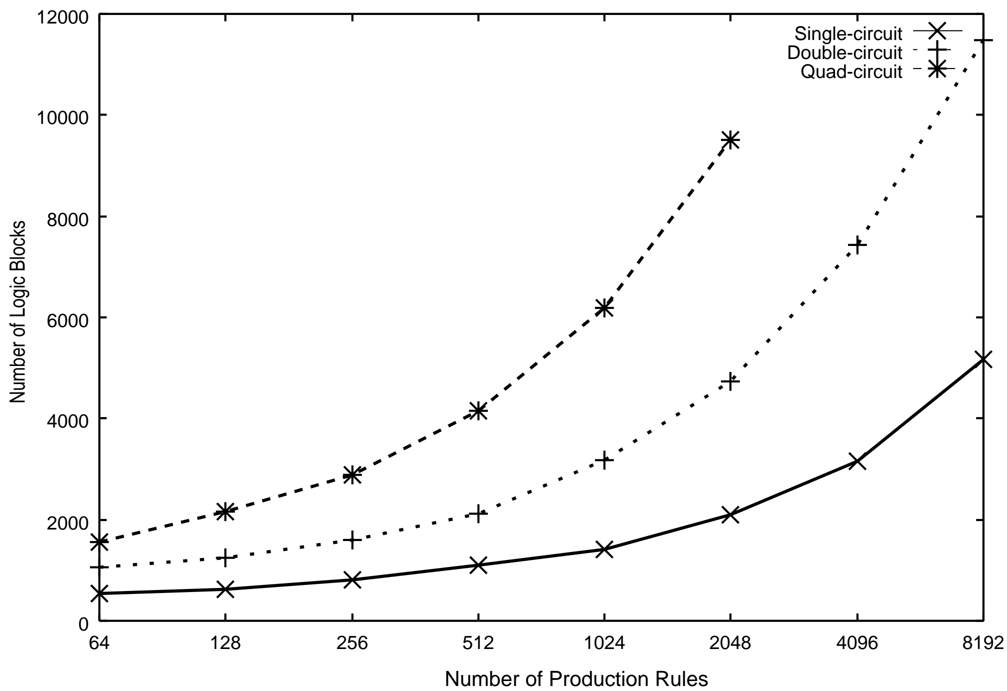

図 4.7: Number of logic blocks used to compute the CKY algorithm with $b = 64$

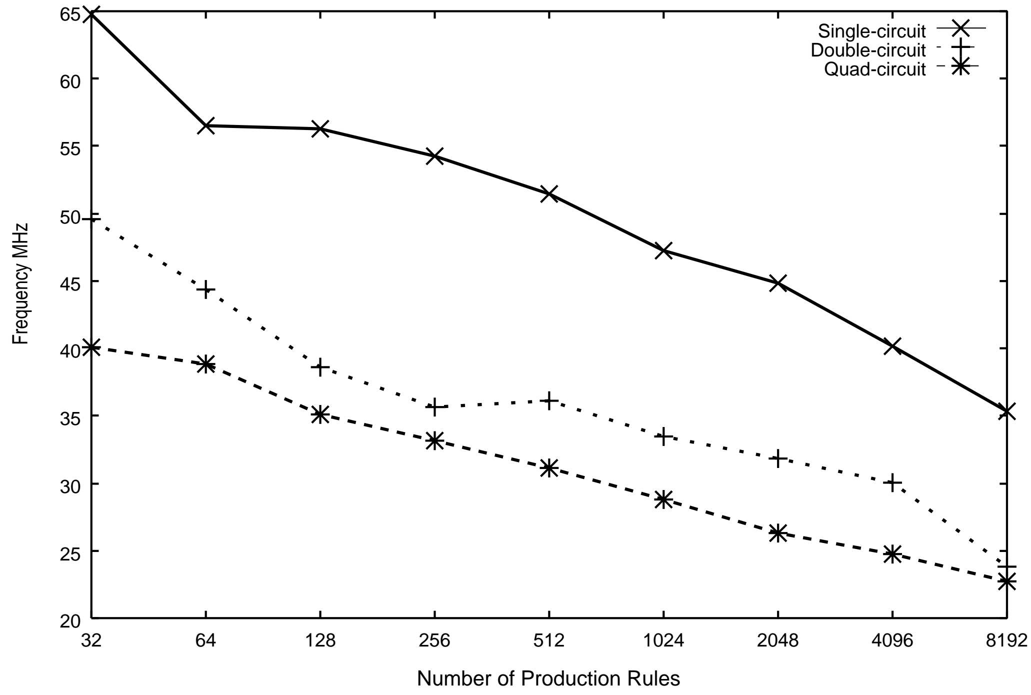

図 4.8: Frequency for single-, doble-, quad-circuit with $b = 32$

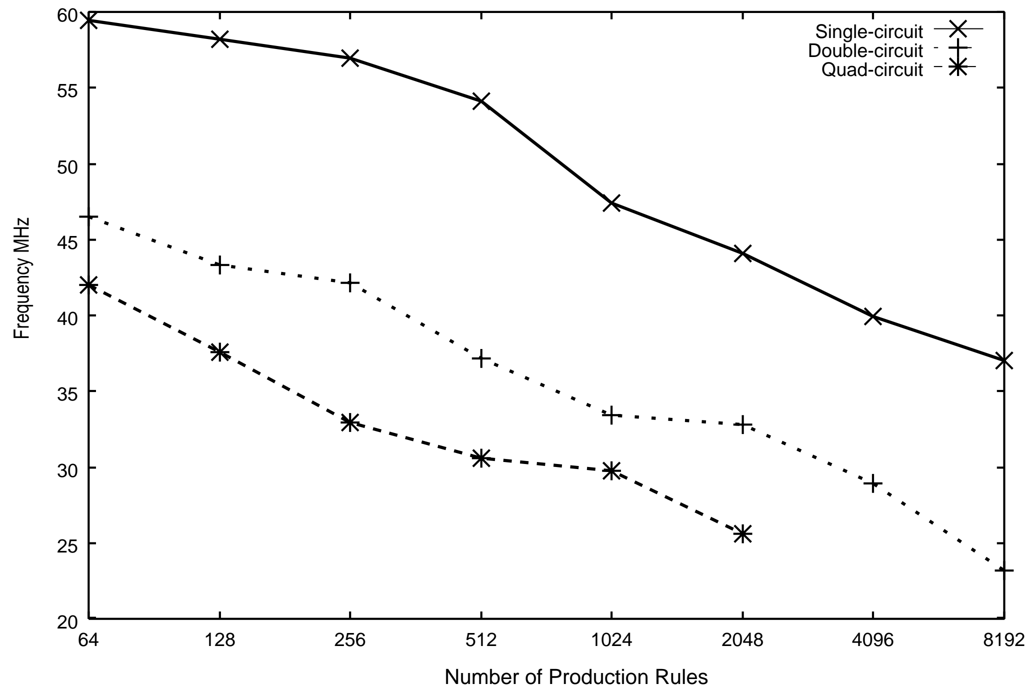

図 4.9: Frequency for single-, doble-, quad-circuit with $b = 64$

第5章　実用的なCKYパージングの実装

4.4 節で述べたように, 前章で実装したハードウェアは入力文字列長や非終端記号数や生成規則数が現実的なサイズではなかった. そこで本章では実用的なサイズで CKY パージングを行うために, 大規模な FPGA を用いたり, 新たなハードウェア実装法の提案を行う.

5.1　ソフトウェアアプローチ

実用的なサイズで CKY パージングを行う際, 3.1.2 節で示した Table アルゴリズムを用いるとテーブルのサイズが巨大になり, このアルゴリズムは使えないことがわかる. このことより, 本章ではソフトウェアアプローチに 3.1.1 節で説明した Naive アルゴリズムを用いることにする.

5.2　ハードウェアアプローチ

5.2.1　ハードウェア実装1

4 章で説明した実装では, 比較的規模の小さい FPGA を用いたので入力文字列が長く非終端記号の数が多い文法では回路規模が大きくなり実装困難であった. そこで 4 章で用いた FPGA よりも大規模な Xilinx 社の Virtex-II シリーズの FPGA を用いて CKY パージングを行う回路を実装した. 回路自体は 4.1 節で説明したハードウェアと同じである. これにより, 入力文字列数 90, 非終端記号数 256, 生成規則数 15,000 の CKY パージングを行う回路を実現した. 今後このハードウェアを**ハードウェア実装** *1* と呼ぶ. さらに大規模な CKY パージングを行うハードウェア実装法を次節で説明する.

5.2.2　ハードウェア実装2

5.2.1 節で説明した実装法は, CKY テーブルを FPGA 内部に持つ. しかし, 入力文字列が長い場合や非終端記号の数が多い文法の場合に CKY テーブルのサイズが巨大になるので, 現在ある FPGA では実装困難である. そこで CKY テーブルをホスト PC のメモリ上に持つことによって, 入力文字列が長い場合や非終端記号の数が多い文法の場合でも実装可能な CKY パージング回路を作成した. この実装は PC と FPGA 間のデータのやり取り

に PCI バスを用い, 回路は PCI バスの動作周波数である 33MHz に同期して動作する．今後, 本節で説明する CKY テーブルを PC のメモリ上にもつ回路を**ハードウェア実装** *2* と呼ぶことにする．このハードウェア実装 2 の基本的な構成要素を次に示す．

– b ビット n ワード (デュアルポート) メモリ

– $\otimes_G$ を計算する CKY 回路

– b 個の OR ゲート列

– b ビットレジスタ

– PC・FPGA 間のインターフェイス回路

b ビット n ワードメモリには, CKY テーブルの 1 行分が記憶される．つまり, CKY テーブルの j 行目 $T[1,j], T[2,j], \ldots$ が記憶される．b ビットレジスタは, CKY パージングの 6 行目で計算される $T[i,j]$ が格納される．b 個の OR ゲート列は, 6 行目の "$\bigcup$" を計算するために用いられる．PC・FPGA 間のインターフェイス回路は, PC と FPGA のデータを PCI を通じてやり取りする際のインターフェイス回路である．このハードウェア実装 2 の回路は, 基本的には CKY テーブル 1 行分を計算する回路である．この回路を用いて CKY テーブルの j 行目を計算する手順を次に示す．

1. CKY テーブル j 行目つまり $T[1,j], T[2,j], \ldots, T[j,j]$ を計算するために必要な CKY テーブルの 1 行目から $j-1$ 行目までの内容を適切な順番に PC から PCI バスを通じて FPGA に送信する．

2. FPGA は, 送信されたデータ順に計算を行い, j 行目の CKY テーブルの内容を b ビット n ワードメモリに格納する．

3. PC 側からすべてのデータの送信が終ったら b ビット n ワードメモリの内容を PC に送信し, PC のメモリ上にある CKY テーブルの j 行目の内容を更新する．

この手順を CKY テーブルの 1 行目から n 行目まですることによって CKY パージングを行う．ハードウェア実装 2 の計算時間は, ハードウェア実装 1 と同じで $O(n^3 \log b)$ である．しかしながら, ハードウェア実装 2 は, ハードウェア実装 1 と比べて CKY テーブルの各行を計算する度に PCI バスを通じて PC と FPGA 間の通信を行う必要があるので, 余分なオーバーヘッドが存在する．よって実行時間はこのオーバーヘッドの分ハードウェア実装 2 の方が長くなる．しかし, FPGA の内部に CKY テーブルを持たないことによって回路規模が小さくなり, 入力文字列が長く非終端記号数が多い CKY パージング回路を実現することが可能となる．

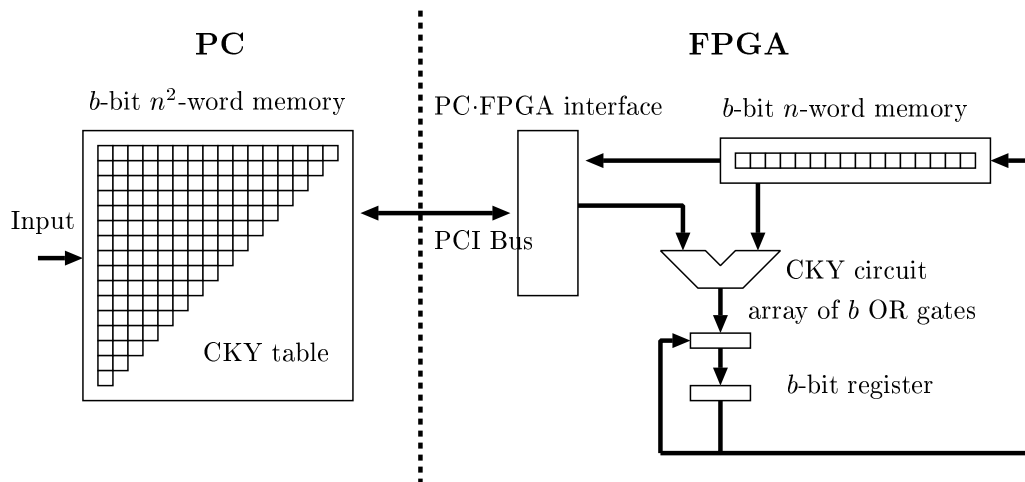

図 5.1: Hardware implementation of the CKY parsing where the CKY table is stored in the host PC.

5.3 性能評価

本節では, 4.3 節で用いた FPGA より大規模な Vertex-II シリーズの FPGA(XC2V3000FG676, 1.7M ビットの組込みメモリと 32K のロジックセルがある 3M ゲート相当の FPGA) を用いてハードウェア実装 1・2 の性能を評価した. 性能評価のために, ハードウェア実装 1,2 とソフトウェアの実行時間の比較を行った. ハードウェア実装では, 次に示す環境で実行し実行時間を計測した.

CPU: Intel Xeon 2.4GHz

メモリ: RDRAM 2GB

OS: Microsoft Windows 2000 Server

コンパイラ: Microsoft Visual C++ 6.0

また, ソフトウェア実装では次に示す環境で実行し実行を計測した.

CPU: Intel Xeon 2.8GHz (FSB 533MHz)

メモリ: PC2100 デュアルチャンネル DDR メモリ 2GB

OS: Linux Kernel 2.2.18-14smp

コンパイラ: Intel C++ Compiler for Linux 7.0

表 5.1: Performance evaluation for the CKY algorithm (the hardware implementation1 in FPGA)

p	FPGA[ms]	Software[ms]	Speed-up
5,000	3.109	2,850	917
10,000		6,050	1,946
15,000		9,100	2,927

表 5.2: Performance evaluation for the CKY algorithm (the hardware implementation2 in FPGA)

n	p	FPGA[s]	Software[s]	Speed-up
128	15,000	0.476	23.54	49
	20,000		33.49	70
	25,000		42.84	90
256	15,000	3.819	119.05	31
	20,000		276.47	72
	25,000		356.23	93
512	15,000	30.456	1,653.95	54
	20,000		2,246.00	74
	25,000		2,823.00	93

非終端記号数 b と入力文字列長 n が $b = 256, n = 90$ で CKY パージングを行ったときの CKY テーブルを FPGA 内に持つハードウェア実装 1 とソフトウェアの実行時間とスピードアップ率を表 5.1 に示す．規則数 p が 5,000, 10,000, 15,000 のランダムに生成した生成規則を用いた．FPGA は生成規則数に依存せず実行時間は一定であることに注意する．また, ソフトウェアアプローチの実行時間は生成規則の数に比例するので, 生成規則数が多くなるにつれて実行時間が長くなっていることがわかる．ハードウェアはソフトウェアに比べ, だいたい 920～3,000 倍の高速化がなされていることがわかる．

非終端記号数 b が $b = 512$ のときのハードウェア実装 2 とソフトウェアの CKY パージングの実行時間を表 5.2 に示す．この表は, 入力文字列長 n を 128, 256, 512, 生成規則数を 15,000, 20,000, 25,000 と変化させたときの実行時間である．入力文字列長や生成規則数が増えるに従ってソフトウェアの実行時間が長くなっていることがわかる．しかし, ハードウェア実装では入力文字列長だけに実行時間が依存するので, 生成規則数が変わっても実行時間は変化しないことに注意する．スピードアップ率より, ハードウェアはソフトウェアに比べ 31～93 倍高速に実行していることがわかる．

表 5.1, 5.2 よりハードウェア実装 1 と比べてハードウェア実装 2 のスピードアップ率が小さいことがわかる．これは, ハードウェア実装 2 は PCI バスを通じてデータのやり取りを行う際のオーバーヘッドがあるからである．

5.4 まとめ

入力文字列長が 90, 非終端記号数 256, 生成規則数 15,000 のサイズで CKY パージングを行うハードウェアを実装し, 4 章で用いた FPGA よりも大規模な Virtex-II シリーズの FPGA を用いて動作を確認した．また, その結果よりソフトウェアと比べて少なくとも約 920 倍, 最大で約 3,000 倍の高速化に成功した．

さらに大規模な CKY パージングを行うために, CKY テーブルをホスト PC メモリ上にもつのハードウェア実装 2 を提案した．入力文字列長 512, 非終端記号数 512, 生成規則数 25,000 のサイズで CKY パージングを行うハードウェアを実装し Virtex-II シリーズの FPGA での動作を確認した．その結果ソフトウェアより最大で約 93 倍の高速化に成功した．

第6章　おわりに

6.1　本研究の成果

本研究では, CKY パージングを行うハードウェアを実装し, FPGA で動作を確認した. 実用的なサイズで CKY パージングを実行するのに, ソフトウェアと比べて 920〜3,000 倍の高速化に成功した. それよりさらに大規模な CKY パージングを実行する場合のハードウェア手法を実装し, 最大で約 93 倍の高速化にも成功した.

6.2　今後の課題

日本語や英語等の実際の言語のパージングを行う問題に対して, 本研究の手法が有効かどうか確かめたい.

謝辞

本研究を進めるにあたり, 終始御指導して下さった中野浩嗣助教授に心より感謝いたします. ゼミなどで大変熱心なご指導をして下さった浅野哲夫教授, 小保方幸次助手, 元木光雄助手に深く感謝致します. 多くの貴重な御意見を頂いた博士後期課程の Jacir Luiz Bordim さんに感謝致します. 最後に, 心の支えになって下さった情報基礎学講座の皆さんに厚く御礼申し上げます.

参考文献

[1] A. V. Aho and J. D. Ullman. *The Theory of Parsing Translation and Compiling.* Prentice Hall, 1972.

[2] Altera Corporation, APEX 20K Devices: System-on-a-Programmable-Chip Solutions, http://www.altera.com/products/devices/apex/apx-index.html.

[3] Altera Corporation. Quartus II:system-on-a-programmable chip sowtfware. http://www.altera.com/products/software/quartus2/qts-index.html

[4] J. Chang, O. Ibarra, and M. Palis. Parallel parsing on a one-way array of finite-state machines. *IEEE Transactions on Computers*, C-36(1):64–75, 1987.

[5] E. Charniak. *Statistical Language Learning.* MIT Press, Cambridge, Massachusetts, 1993.

[6] C. Ciressan, E. Sanchez, M. Rajman, and J.-C. Chappelier. An FPGA-based coprocessor for the parsing of context-free grammars. In *Proc. of IEEE Symposium on Field-Programmable Custom Computing Machines*, 2000.

[7] C. Ciressan, E. Sanchez, M. Rajman, and J.-C. Chappelier. An FPGA-based syntactic parser for real-life almost unrestricted context-free grammars. In *Proc. of International Conference on Field Programmable Logic and Applications (FPL)*, pages 590–594, 2001.

[8] Y. Futamura, K. Nogi, and A. Takano. Essence of generalized partial computation. *Theoretical Computer Science*, 90:61–79, 1991.

[9] A. Gibbons and W. Rytter. *Efficient Parallel Algorithms.* Cambridge University Press, 1988.

[10] N. D. Jones, C.K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation.* Prentice Hall, 1993.

[11] S. R. Kosaraju. Speed of recognition of context-free languages by array automata. *SIAM J. on Computers*, 4:331–340, 1975.

[12] J. C. Martin. *Introduction to languages and the theory of computation (2nd Edition)*. Mac-Graw Hill, 1996.

[13] Takashi Ninomiya, Kentaro Toriswa, Kenjiro Taura, and Jun-ichi Tsujii. A Parallel CKY Parsing Algorithm on Large-Scale Distributed-Memory Parallel Machines, in *Proc. of the Pacific Association for Computational Linguistics '97*, pp. 223–231, Tokyo, Japan, September, 1997.

[14] Y. Sakakibara, M. Brown, R. Hughey, I. S. Mian, K. Sjölander, R. C. Underwood, and D. Haussler. Stochastic context-free grammars for tRNA modeling. *Nucleic Acids Research*, 22:5112–5120, 1994.

[15] M. P. van Lohuizen. Survey on parallel context-free parsing techniques. Technical Report IMPACT-NLI-1997-1, Delft University of Technology, 1997.

[16] Xilinx Corporation, Virtex-II FPGAs, http://www.xilinx.co.jp/

[17] Xilinx Corporation. ISE ロジック デザインツール. http://www.xilinx.co.jp/ise/design_tools/index.htm

論文リスト

1. J. L. Bordim, Y. Ito, and K. Nakano, "Accelerating the CKY Parsing Using FPGAs", to appear in the IEICE Transactions on Information and Systems.

2. J. L. Bordim, Y. Ito, and K. Nakano, "Accelerating the CKY Parsing Using FPGAs", Proc. of the 9th International Conference on High Performance Computing (HiPC-2002), Dec. 18-21, Bangalore, India.

3. 伊藤 靖朗, "FPGA を用いた CKY 法による構文解析の高速化", 平成 14 年度 電気関係学会北陸支部連合大会 講演論文集 , pp. 217, September 2002.

4. Y. Ito, J. L. Bordim, and K. Nakano, "Accelerating the CKY Parsing using FPGAs", Technical Report of IPSJ, AL2002-85 , pp. 35-42, July 2002.

Appendix

ここでは, 簡単な文脈自由文法の例を用いて CKY パージングを行ったときのデータを示す.

1. $x^n y^n$
   次のような文脈自由言語で表される文脈自由文法を実装した.

$$\{x^n y^n | x \in \Sigma, y \in \Sigma, n \geq 1\}$$

   $|\Sigma| = 2$ のときの文脈自由文法 $G = \{N, \Sigma, P, S\}$ は次のようになる.

$$\begin{aligned}
&N = \{S, A, B\}, \\
&\Sigma = \{a, b\}, \\
&P = \{S \rightarrow AB, \\
&\qquad S \rightarrow AS_1B, \\
&\qquad S_1 \rightarrow AS_1B, \\
&\qquad S_1 \rightarrow AB, \\
&\qquad A \rightarrow a, \\
&\qquad B \rightarrow b\}
\end{aligned}$$

   今回はアルファベットが 9 文字, $n = 45$ のときの CKY パージング回路を作成した. このとき生成規則数 486, 非終端記号数 253 であった. ハードウェア実装 1 とソフトウェアの実行時間スピードアップ率を次に示す.

FPGA[ms]	Software[ms]	Speed-up
3.109	170.0	55

2. 回文

次のような文脈自由言語で表される文脈自由文法を実装した.

$$\{x \in \Sigma^* | x = x^{\mathrm{R}}\}$$

$|\Sigma| = 2$ のときの文脈自由文法 $G = \{N, \Sigma, P, S\}$ は次のようになる.

$$
\begin{aligned}
N &= \{S, A\}, \\
\Sigma &= \{a, b\}, \\
P &= \{S \to aAa, \\
&\quad\; S \to bAb, \\
&\quad\; S \to a, \\
&\quad\; S \to b, \\
&\quad\; A \to a, \\
&\quad\; A \to b,
\end{aligned}
$$

今回はアルファベットが 127 文字で入力文字列長が 90 文字のときの CKY パージング回路を作成した. このとき生成規則数 381, 非終端記号数 255 であった. ハードウェア実装 1 とソフトウェアの実行時間とスピードアップ率を次に示す.

FPGA[ms]	Software[ms]	Speed-up
3.109	140.0	45