博士学位論文

情報圧縮のための並列処理アーキテクチャに関する研究

東北大学大学院工学研究科
電気・通信工学専攻

藤岡 豊太

目 次

第 1 章 序論 **1**

1.1 背景： データ符号化・復号化技術の概要、その重要性 1

1.2 データ符号化・復号化技術の現状 1

1.3 本研究の目的 4

1.3.1 データ圧縮技法のハードウェア化の現状 4

1.3.2 ハードウェア化の方向性 5

第 2 章 LZ77 符号と出力符号の統計的特徴 **6**

2.1 Lempel-Ziv 符号の概念 6

2.2 LZ77 符号 ~ LZ77 符号化 ~ 7

2.2.1 最長一致記号系列探索 8

2.2.2 LZ77 符号化アルゴリズム 9

2.2.3 LZ77 符号化の特徴 11

2.3 LZ77 符号 ~ LZ77 復号化 ~ 11

2.3.1 LZ77 復号化アルゴリズム 11

2.3.2 LZ77 復号化の特徴 15

2.4 LZ77 符号の改善手法 15

2.4.1 LZSS 符号 15

2.4.2 LZB 符号 16

2.4.3 LZH 符号 17

2.4.4 LZBW 符号 17

2.5 LZ77 符号の統計的特徴 18

2.5.1 LZ77 符号化での最長一致記号系列探索 18

2.5.2 生成符号の統計的特徴を利用した計算量の削減 18

2.5.3 1 データファイルあたりの最長一致記号系列長の頻度 ~ 一符号あたりに必要な比較演算回数 ~ 20

2.5.4 一符号あたりの平均比較回数 21

2.5.5 符号の統計的特徴を利用した場合の計算量 34

第 3 章 高速 LZ77 符号化・復号化並列処理アーキテクチャ - PAHL **36**
3.1 高速 LZ77 符号化並列処理アーキテクチャ - PAHL-C 36
3.1.1 提案アーキテクチャに必要な機能 36
3.1.2 高速 LZ77 符号化並列処理アーキテクチャ - PAHL-C の基本構成 . 37
3.1.3 PAHL-C の各計算セルとその機能 40
3.1.4 PAHL-C の計算量 43
3.1.5 PAHL-C の性能評価 44
3.1.6 PAHL-C の論理合成結果 46
3.1.7 PAHL-C の実装における課題 52
3.2 高速 LZ77 復号化並列処理アーキテクチャ - PAHL-D 52
3.2.1 LZ77 復号化アルゴリズム内の並列性 52
3.2.2 PAHL-D の基本構成 53
3.2.3 PAHL-D の各機能 54
3.2.4 PAHL-D の性能評価 55
3.2.5 PAHL-D の実装への課題 58
3.3 高速 LZ77 符号化・復号化並列処理アーキテクチャ - PAHL 59
3.3.1 符号化・復号化アーキテクチャの統合による利点・課題 59
3.3.2 LZ77 符号化・復号化アルゴリズムの共通点 59
3.3.3 PAHL-C、PAHL-D の共通点 60
3.3.4 高速 LZ77 符号化・復号化並列処理アーキテクチャ - PAHL の基本構成 61
3.3.5 PAHL の性能評価 63
3.3.6 PAHL の論理合成結果 64
3.3.7 PAHL の予測性能 64

第 4 章 高速 LZSS 符号化・復号化並列処理アーキテクチャ - PAHL-LZSS **67**
4.1 LZSS 符号 67
4.1.1 LZSS 符号化 67
4.1.2 LZSS 復号化 69
4.2 PAHL から PAHL-LZSS への拡張 70
4.3 高速 LZSS 符号化並列処理アーキテクチャ - PAHL-LZSS-C 70
4.3.1 PAHL-LZSS-C の基本構成 70
4.3.2 PAHL-LZSS-C の性能評価 74
4.4 高速 LZSS 復号化並列処理アーキテクチャ - PAHL-LZSS-D 77
4.4.1 PAHL-LZSS-D の基本構成 77
4.4.2 PAHL-LZSS-D の性能評価 80
4.5 高速 LZSS 符号化・復号化並列処理アーキテクチャ - PAHL-LZSS 83
4.5.1 PAHL-LZSS の基本構成 83
4.5.2 PAHL-LZSS の性能評価 85

第 5 章 モジュール化による PAHL の実装 **89**
5.1 PAHL、PAHL-LZSS の実装における問題 89
5.2 PAHL-C の実装 ~ モジュール化 ~ 90
5.2.1 モジュール化のための分割法 90
5.2.2 各機能モジュール 90
5.3 各機能モジュールによる PAHL-C の構成 94

第 6 章 結論 **96**
6.1 はじめに 96
6.2 高速 LZ77 符号化・復号化並列処理アーキテクチャ - PAHL 97
6.3 高速 LZSS 符号化・復号化並列処理アーキテクチャ - PAHL-LZSS 99
6.4 PAHL の実装 ~ モジュール化 ~ 100
6.5 おわりに 100

謝辞 **102**

参考文献 **103**

研究業績 **105**

図 目 次

1.1 data-codec-category 3

2.1 slide-window 8
2.2 dictionary-ref 8
2.3 lz77-codec-proc 12
2.4 lz77-decodec-proc 14
2.5 My-LMS-Search 19
2.6 length-ratio-2B 22
2.7 length-ratio-C1 22
2.8 length-ratio-C2 23
2.9 length-ratio-P1 23
2.10 length-ratio-P2 24
2.11 length-ratio-R1 24
2.12 length-ratio-T1 25
2.13 length-ratio-T2 25
2.14 length-ratio-T3 26
2.15 length-ratio-T4 26
2.16 length-ratio-T5 27
2.17 length-ratio-U1 27
2.18 average-comp-2B 28
2.19 average-comp-C1 29
2.20 average-comp-C2 29
2.21 average-comp-P1 30
2.22 average-comp-P2 30
2.23 average-comp-R1 31
2.24 average-comp-T1 31
2.25 average-comp-T2 32
2.26 average-comp-T3 32
2.27 average-comp-T4 33
2.28 average-comp-T5 33
2.29 average-comp-U1 34

3.1 PAHL-C-Struct 39
3.2 cell1-circuit 41
3.3 cell2-circuit 42
3.4 all-match-circuit 43
3.5 const-order 45
3.6 PAHL-C-Power 46
3.7 PAHL-C-Area 48
3.8 PAHL-C-Gate 49
3.9 PAHL-C-Cells-Delay 51
3.10 PAHL-D-Struct 54
3.11 PAHL-D-Power 55
3.12 PAHL-D-Area 56
3.13 PAHL-D-Gate 56
3.14 PAHL-D-Compo–Delay 57
3.15 PAHL-module 60
3.16 PAHL-Struct 62
3.17 PAHL-Power 65
3.18 PAHL-Area 65
3.19 PAHL-Gate 66

4.1 lzss-codeform 69
4.2 PAHL-LZSS-C-Struct 72
4.3 PAHL-LZSS-C-Power 75
4.4 PAHL-LZSS-C-Area 76
4.5 PAHL-LZSS-C-Gate 76
4.6 PAHL-LZSS-D-Struct 78
4.7 PAHL-LZSS-D-Power 81
4.8 PAHL-LZSS-D-Area 81
4.9 PAHL-LZSS-D-Gate 82
4.10 PAHL-LZSS-D-Delay 82
4.11 PAHL-LZSS-Struct 84
4.12 PAHL-LZSS-Power 86
4.13 PAHL-LZSS-Area 86
4.14 PAHL-LZSS-Gate 87

5.1 PAHL-module-cell1 91
5.2 PAHL-module-cell2 92
5.3 PAHL-module-count 93
5.4 PAHL-module-1k 95

表 目 次

2.1 LZ77 符号の改良符号 15
2.2 サンプルデータ 21
2.3 各データファイルにおける平均比較回数 28

3.1 PAHL-C の消費電力 47
3.2 PAHL-C の実装面積 47
3.3 PAHL-C のゲート数 48
3.4 PAHL-C の予測性能 51
3.5 PAHL-D の消費電力、実装面積、ゲート数 57
3.6 PAHL-D の予測性能 58
3.7 予測される PAHL の性能及び PAHL-C、PAHL-D との比較 66

4.1 PAHL-LZSS-C の予測性能 77
4.2 PAHL-LZSS-D の予測性能 83
4.3 予測される PAHL-LZSS の性能及び PAHL-LZSS-C、PAHL-LZSS-D との比較 88

5.1 一致長探索モジュールの性能 (ref. buffer : 128, cell1 : 128) 91
5.2 最長一致選択モジュールの性能 (cell2 : 127) 92
5.3 カウント module の性能 93
5.4 module 化による PAHL-C の構成 (ref. buffer : 1K[byte]) 95

第 1 章

序論

1.1 背景： データ符号化・復号化技術の概要、その重要性

コンピュータ技術が大きく発達し、それと共にコンピュータが一般に広く普及している現在において、コンピュータで利用される情報 (以下、データと呼ぶ) は大量且つ多種多様となっている。LAN、WAN、Internet 等などの各種 Network などでは、様々なデータの送受信 (伝送) が頻繁に行なわれ、同時に多種多様なデータ管理・蓄積が行われる。以上の状況におけるデータの取り扱いにおいて重要な要素となっているのが、いかにデータを高速にやり取りできるか、また効率的に管理、蓄積しておくことができるかという点になる。高速なデータ伝送、蓄積資源の効率的な活用は、リアルタイムなデータ処理、効率的なデータ運用に大きな役割をはたす。

効率的なデータの伝送や蓄積・管理のための重要な技術がデータ圧縮 (符号化・復号化) 技術である。データを圧縮することにより、少ない蓄積資源をより有効に活用することが可能となり、データ伝送時間の短縮や単位時間あたりのデータ伝送量の増加が可能となる。

1.2 データ符号化・復号化技術の現状

データ圧縮の起源は 18 世紀までさかのぼることができる。初期のデータ圧縮は、主に通信時間の短縮や効率化の目的に用いられているもので、この中で、現在でも利用されている最も有名なものが、S. Morse が 1832 年に電信用に発明したモールス信号である。

データ符号化・復号化技術は、大きく次の二種類に分類される [13]。

(1) 可逆 (Lossless) 圧縮

(2) 非可逆 (Lossy) 圧縮

(1) は、圧縮前の元データと圧縮したものの復元後のデータが、まったく同一となるような圧縮のことである。テキスト文書や実行形式ファイル等では、それらを圧縮したファイルから復元して得られたファイルが、圧縮前と同一でなければ、復元後の文書の内容が書き変ったり、実行ファイルが正常に実行されない等の不具合が生じる。そのため、完全に元のデータを復元することが可能である圧縮 (符号化) を可逆圧縮 (reversible compression、lossless coding) と呼ぶ。

(2) は、例えば画像、音声データなどでは、圧縮したデータから復元したデータが必ずしも完全に同一なものである必要がなく、元データとの若干の差異を許容することで、圧縮後のデータ容量をより小さくすることが可能となる。このように、若干の雑音が実用上影響を与えないようなデータ (画像、音声データなど) において、圧縮前の元データと復元後のデータの若干の差異を許容するとによって、より効率的なデータ圧縮を実現する圧縮方式を非可逆圧縮 (irreversible compression、lossy coding) と呼ばれる。

一般に、非可逆圧縮のほうが可逆圧縮よりも大幅なデータの圧縮が可能であるが、その使用は画像、音声などのデータに限られ、それらのデータの持つ特徴を利用した、データに特化された圧縮法である場合が多い。そこで本論文では、以後可逆圧縮について議論する。

可逆圧縮の代表的な手法としては、次にものが存在する。

- シャノン・ファノ符号 (Shannon and Fano, 1948)
- Huffman 符号 (Huffman, 1952)
- 算術符号 (Elias, 1960 頃)
- LZ77 符号、LZ78 符号 (Ziv and Lempel, 1977,78)
- 適応型 Huffman 符号
- 適応型算術符号

シャノン・ファノ符号、Huffman 符号、算術符号は、始めに情報の出現頻度などから情報源のモデルを作成し、それに基づいて情報を符号化していく。この方式では、符号化において常にモデルは変化することはなく、記号と符号語の対応が常に一定である。これらの中で代表的な符号化法となるのが Huffman 符号である。

LZ77、LZ78、適応型 Huffman 符号、適応型算術符号などは、入力記号列の特性の変化に適応できるという特徴を持つ。これらの中には、入力される記号列の特性によらず、どのような情報源から生成された記号列に対しても、その記号列が長くなるにつれて、最良の圧縮率が達成できる万能なデータ符号化がある。このような符号化は特にユニバーサル符号化 (universal coding) と呼ばれる。このユニバーサル符号化の代表的なものが Lempel-Ziv 符号化 (LZ77 符号、LZ78 符号) である (図 1.1)。

データ圧縮に関する近代的な研究は、1940 年代後半からシャノンによって開始され、1950 年代になると、D.A.Huffman が、各記号の生起確率が与えられたとき、平均的な伝送時間が最小になるような符号の構成法を提案し、一般の記号列に対してもモールス符号と同様

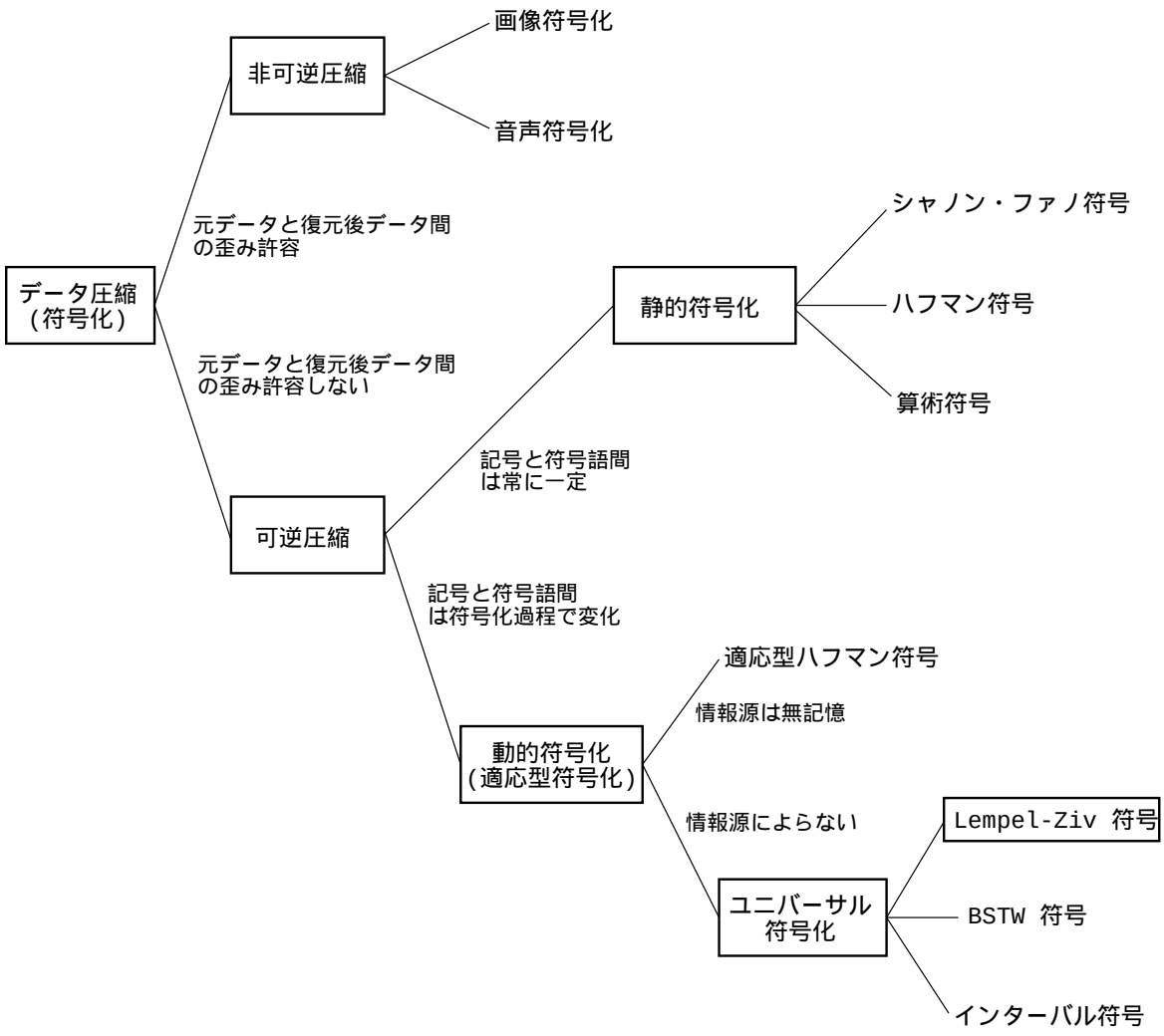


図 1.1: データ圧縮技法の分類

の性質を持つ符号が自由に作られるようになった。また 1970 年代のマイクロプロセッサの発明ににより、データ圧縮システムを実現するハードウェアの基盤がつくられた。

現在では、メモリやハードディスクの低価格化と普及に伴い、実際にデータ圧縮技術がいろいろな分野で要求されるようになってきている。しかし、現在データ圧縮が普及した理由としては、実用的なデータ圧縮アルゴリズムの開発、特に J.Ziv と A.Lempel が 1977 年及び 1978 年に発明した二種類のデータ圧縮アルゴリズム - LZ77 法、LZ78 法に負うところが大きいものと考えられる。実際、現在実用化されているデータ圧縮ソフトウェアの大部分はこのアルゴリズムを応用している。

現在データ圧縮技術が様々な分野で用いられているが、これらデータ圧縮アルゴリズムの基盤となっている考えは、モールスが 1838 年に考えた、出現頻度の高い記号は短い符号語で、出現頻度の低い記号は長い符号語で表現する、という理論に基づいている。この意味から、データ圧縮分野はまだ発展を残している分野であるものと考えられる。

1.3　本研究の目的

現在実用化されているデータ圧縮技術は、大部分はソフトウェアとして CPU 上で実行されている。実際にデータ圧縮が用いられるべきシステムの要求を考えた場合、

- データ圧縮速度

  単位時間あたりのデータ圧縮 / 復元量

- データ圧縮の他処理との並行性

  データに対して実行される各処理とデータ圧縮処理との独立実行性

という面から、ソフトウェア的なデータ圧縮法では、システムをより効率的に運用するのに十分ではない。そこで、以上の不十分な点に対応するために、データ圧縮ハードウェア化がデータ圧縮技術の発展に重要である。以上のような状況から本論文では、より効率的なデータ伝送・蓄積において重要な手段となる、高速データ圧縮 (符号化・復号化) アーキテクチャの構築をその目的としている。

1.3.1　データ圧縮技法のハードウェア化の現状

マイクロプロセッサの発明、また近年の VLSI/ULSI 技術の発達により、様々なアルゴリズムが ASIC として構成されるようになってきている。データ圧縮技術もまた例外ではなく、現在までにの各データ圧縮アルゴリズムを実装したデータ圧縮ハードウェアが提案、実装されている [1] [2] [4] [5] [6] [7]。

ハードウェア化の対象とするデータ符号化・復号化アルゴリズムの決定にあたって、以下のことを考慮するものとした。

(1) 高速動作が可能となりうる。

(2) 広範囲の情報源に対し有効である。

(3) 汎用性が高い。

(4) ハードウェアとして実装が容易である。

以上の点を考慮すると、(I) 符号化前にあらかじめ情報源によるモデルを生成する必要がなく、(II) 多種多様な情報源に対し有効であるユニバーサル符号であり、(III) 現在実用化されているデータ圧縮ソフトウェアの大部分に用いられており、(IV) アルゴリズムが比較的単純であるという点から、Lempel-Ziv 符号が対象アルゴリズムとして適当であるものと結論づけた。Lempel-Ziv 符号には大別して

- LZ77 符号
- LZ78 符号

の二種類がある。本論文では、前記の (1) ~ (4) の条件にあてはまるデータ圧縮技法の一つとして、LZ77 符号を対象アルゴリズムとして採用した。

本論文では以後、LZ77 符号化・復号化ハードウェア化について議論する。

1.3.2　ハードウェア化の方向性

データ圧縮技法、特に本論文では LZ77 符号のハードウェア化に対して、次の (1)、(2)、(3) を考慮した。

(1) 高速処理可能な手法

- ハードウェア化の利点を生かし、VLSI/ULSI 技術をによる並列処理技術等を応用し、ソフトウェア的には実現不可能な処理技法をアーキテクチャに取り入れる。

(2) 実装回路の小規模化

- 容易に ASIC として実現可能な、できるだけ簡素で小規模なアーキテクチャを実現する。

(3) 汎用性

- ある特定の処理対象だけではなく、既存の (Lempel-Ziv 符号を応用した) データ圧縮ソフトウェアに容易に応用できるようにする。

第 2 章

LZ77 符号と出力符号の統計的特徴

2.1 Lempel-Ziv 符号の概念

データ圧縮アルゴリズムを大別すると、何らかの形で情報の確率モデルを作成して、出現頻度の大きい記号には短い符号を、出現頻度の小さい記号には長い符号を割り当てることで圧縮を実現するアルゴリズムと、語 (word) や節 (phrase) と呼ばれる記号列を、固定長または可変長の符号語に変換する、いわゆる辞書に基づいた符号化によるアルゴリズムがある。ここで述べる Lempel-Ziv 符号は「辞書に基づいた符号化」手法の一つであり、入力される記号列の統計的特徴によらず、どのような情報源から生成された記号列に対しても、その記号が長くなるにつれて最良の圧縮率が達成できる万能なデータ符号化法 (ユニバーサル符号化) の一つである。

辞書に基づく符号化では、ある記号列 (語、単語) を符号語に対応させることが符号化の基本となる。このとき、符号語のビット長が対応する記号列のビット長よりも短い場合にデータ圧縮が実現される。この符号語のビット長は、どのような辞書を作成するかに依存するものである。すなわち、生成される辞書、またその更新法によって、符号化の性能が変化する [13]。

辞書も基づく符号化法には、大別して

- 静的辞書法 (static dictionary method)
- 適応型辞書法 (adaptive dictionary method)

の二種類がある。

静的辞書法は、符号化にあたりあらかじめ辞書を編集し、符号化を実行している間は辞書の更新を行わない方式である。この手法では、データ符号化にあたり、圧縮するデータに最適な辞書を用意できるという点が最大の利点となる。しかし、符号化と復号化で同じ辞書を用いる必要があり、符号化で用いられた辞書をどのよに復号化部に渡すべきかが問題となる。

一方、適応型辞書法は辞書を前もって編集せず、符号化する記号列を読み込み符号化しながら、順次辞書を編集していく方式である。基本的なアルゴリズムとしては、

(1) データファイルから記号列を読み込む。

(2) 読み込んだ記号列を辞書から探索し、対応する辞書番号 (ポインタ : 例えば非負数) を求める。もし辞書内に記号列が登録されていなければ、辞書番号として任意数 (例えば-1) を返す。

(3) 得られた辞書番号から以下の処理を実行する。
  (3.1) 辞書番号が非負数 (記号列が辞書内に存在) の場合
  符号語として辞書番号の適切な 2 進数表現として出力する。
  (3.2) 辞書番号が -1 (記号列が辞書内に存在しない) の場合
  符号語として、その記号列に適切な 2 進数表現を出力した後、辞書に符号化した記号列を登録する。

(4) データファイル内の全ての記号列を符号化するまで (1) ~ (3) の処理を繰り返す。

となる。この手順の繰り返しにより辞書に登録してある記号列を増加し、符号化が進むにつれてデータが圧縮されるのに十分な記号列が登録されていくことになる。従って、適応型辞書法では、符号化の最初の部分では辞書が不十分なため、データ圧縮がうまくいかず逆にデータ伸張が起きる可能性がある。しかし、十分に長い記号列に対しては、記号列を読も込むにつれて辞書が充実してくるので、優れたデータ圧縮を実行することができる。以上のように適応型辞書法では、十分に長い記号系列に対して効果的な圧縮が得られるものである。

現在最も一般的な適応型辞書法が、イスラエルの情報理論家の J.Ziv と計算機科学家の A.Lempel により提案された Lempel-Ziv 符号化である。この符号化には多種なバリエーションが存在する。本論文では、

- 多種多様な情報に対しての有効性
- 現在利用されているデータ圧縮アプリケーションへの応用

などの条件から、1977 年に発表された LZ77 符号をハードウェア化を行なうデータ圧縮法として採用している。

2.2　LZ77 符号 ~ LZ77 符号化 ~

LZ77 符号では、既に符号化を終えた記号列を辞書として利用することで符号化を実行する。具体的には、現在符号化される記号系列の直前の長さ N のの記号系列を参照部バッファ(辞書バッファ) に、符号化される記号系列を符号化部バッファにそれぞれ入れ、そのバッファ間の最長一致記号系列を探索することで符号化を実現する。このバッファは符号化につれて与えられた記号系列をスライドしていくことになり、このバッファはスライド窓 (バッファ) とも呼ばれる (図 2.1) [3] [13]。

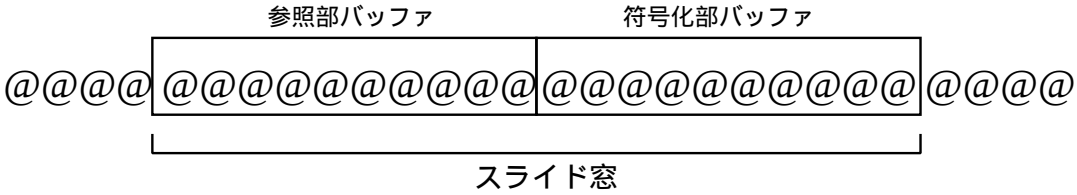


図 2.1: スライド窓バッファ

2.2.1　最長一致記号系列探索

LZ77 符号化における辞書の参照は、現在符号化の対象となっている位置から始まる記号列と一致する、スライド窓内で最長の記号列 (最長一致記号系列) を探索する処理を基本としている。

辞書を示すポインタとして、記号列 ”S” の m 番目から始まる長さ l の記号列 S[m, m+l-1] を (m, l) として表現することにする。例えば (3, 3) とすると、入力記号の 3、4、5 番目の記号から成る記号列であることを意味する。この表現により ”ababbababaa” は、”ab(1, 2)(2, 3)(6, 3)(10, 1)” のように表現することができる (図 2.2)。

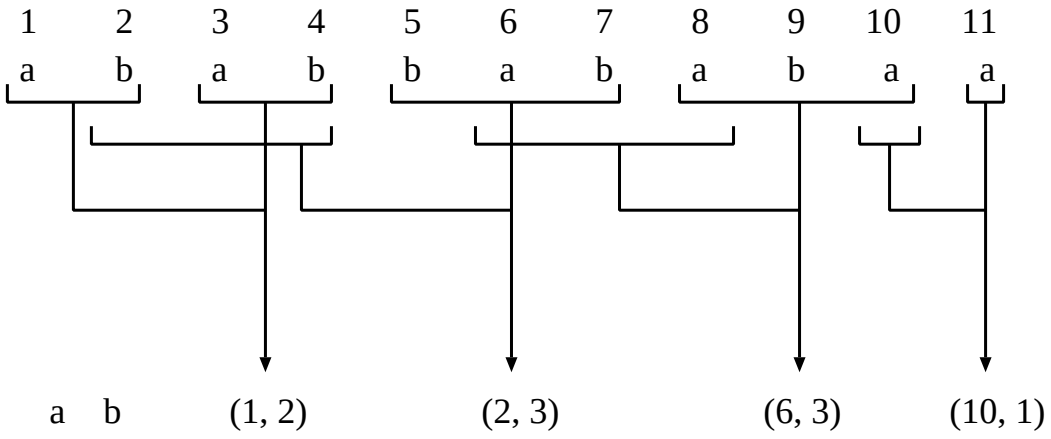


図 2.2: LZ77 符号化における辞書参照の考え方

逆に、”ab(1, 2)(2, 3)(6, 3)(10, 1)” のような表現が与えられたとき、元の記号系列は、ポインタの差し示す記号列によってポインタを置き返ることによって復元することができる。ポインタ (1, 2) の差し示す記号列を ”ab” であり、(1, 2) を復元した時点で記号系列 ”abab(2, 3)(6, 3)(10, 1)” を得る。次にポインタ (2, 3) を復元すると ”ababbab(6, 3)(10, 1)” を得る。次にポインタ (6, 3) を復元するにあたり、この時点ではまだ 8 番目に対応する記号は得られていない。このため、まず 6、7 番目の記号を復元し ”ababbabab” までを確定し、

8 番目が "a" であることが確定された後に、8 番目の記号を復元し"ababbababa(10, 1)" を得る。最後にポインタ (10, 1) を復元し、元の記号系列 "ababbababaa" が復元される。

このような辞書参照法、つまり最長一致記号系列を探索することにより符号化を実行する手法が、LZ77 符号化の基本である。

2.2.2　LZ77 符号化アルゴリズム

実際の LZ77 符号化では、図 2.3に示すように、既に符号化を終えた記号系列を参照部バッファに、現在符号化される記号系列を符号化部バッファに置き、符号化部バッファの先頭から始まる記号系列と、参照部バッファの各位置から始まる記号系列との最長一致記号系列を探索することにより、求められた記号系列の参照部バッファで先頭位置、長さ、また符号化部バッファでの求められた記号系列に続く未一致記号を最適に 2 進数化したものを符号として出力する。この二つのバッファの組をスライド窓と呼ぶ。

以下に LZ77 符号化の基本アルゴリズムを示す。

(1) 初期設定

情報源から得られる記号系列を $(a_0, a_1, a_2, ..., a_{Z-1})$ とする。得られる記号系列内の記号は $\lceil \log_2 Z \rceil$ [bit] で表現することができる。

次に参照部バッファ長を N、符号化部バッファ長を M とする。一般には、N としては 1024 ~ 4096 程度、M としては 16 ~ 128 程度が圧縮率の面で最適であると言われている。参照部バッファ内の初期値として a_0を、符号化部バッファには符号化すべき記号系列の最初の記号系列を置く。このとき、参照部バッファ内の記号列 S、符号化部バッファ内の記号列 B とし、求められる最長一致記号系列の参照部バッファでの先頭位置 P、一致長 L はそれぞれ

$$\mathrm{P} = \lceil \log_2 N \rceil \ [\mathrm{bit}]$$
$$\mathrm{L} = \lceil \log_2 M \rceil \ [\mathrm{bit}]$$

の 2 進数で表現できる。また、現在符号化を行っている記号の入力記号系列中での位置を示すポインタ i を $\mathrm{i} = 0$ とする。

(2) i 番目の記号から始まる記号列の符号化

i 番目から始まる記号列に対する最長一致記号系列を参照部バッファから探索する。つまり、ある k $(0 \leqq \mathrm{k} < \mathrm{M})$ に対して

$$\mathrm{S[m,\ m+k\text{-}1]} = \mathrm{B[i,\ i+k\text{-}1]}$$

を満す k $(0 \leqq \mathrm{k} < \mathrm{m})$ の中で最長のものを探索する。もし見つからなければ $\mathrm{m} = 0$、$\mathrm{k} = 0$ とする。

(3) 符号の出力

中間符号 (P, L, W) を次のように決定する。

P：最長一致記号系列の参照部バッファでの先頭からの位置

L：最長一致記号系列の長さ

W：符号化部バッファでの一致しなかった最初の記号

実際の符号語は、P、L、W をそれぞれ最適に 2 進数化したものを順に並べたものとなる。

(4) スライド窓の更新

スライド窓の位置を L+1 記号だけ記号列の後方へずらす。つまりスライド窓 (参照部バッファ) の先頭 L+1 記号を捨て、新たに L+1 の記号列を符号化部バッファに追加する。また、符号化位置を示すポインタを

$$i \leftarrow i + L + 1$$

とした後、(2) へ戻る。

次に N=4、M=4 の場合について、"ababbababaa" という記号系列を符号化する具体的な過程を、上記のアルゴリズムに沿って示す (図 2.3)。

(1) 初期化

参照部バッファ内は記号 "a" で初期化され、符号化部バッファには符号化すべき記号系列の最初の 4 記号 "abab" が置かれる。

また、$i = 1$ としておく。

(2) 最長一致記号系列探索 (符号化)

$i = 1$ からはじまる記号系列に対する符号化のため、参照部バッファと符号化部バッファ間の最長一致記号系列探索によって、長さ 1 の記号列"a" が得られる。よって中間符号として、一致記号列の先頭位置となる 0 (同じ長さの一致記号系列が探索された場合、最も先頭に近いものを採用する)、一致長である 1、未一致記号となる "b" を並べて〈0, 1, "b"〉を中間符号とする。

(3) 符号の出力

(2) で得られた中間符号を最適な 2 進数表示に変換したものを符号として出力する。

(4) スライド窓の更新

スライド窓 (参照部バッファ + 符号化部バッファ) 内の記号系列を 2 記号分ずらし、新たに符号化部バッファに "ba" を追加する。また $i = 1 + 2 = 3$ とし、次の符号化を行う。

(5) 最長一致記号系列探索 (符号化)

次に、i = 3 から始まる記号系列についての符号化が行なわれる。(2) と同様の作業で求められた最長一致記号系列は "ab" となる。従って得られる一致記号系列の先頭位置 2、一致長 2、未一致記号 "b" より、中間符号は〈2, 2, "b"〉となる。

その後、スライド窓の記号を 3 記号ずらし、符号化部バッファに新たに "bab" を追加する。また i = 3 + 2 + 1 = 6 とし、(2) と同様に符号化を継続する。

以下、符号化部バッファ内の記号が全て符号化されるまで続けられる。

2.2.3　LZ77 符号化の特徴

LZ77 符号化においては、最長一致記号系列探索により所望の最長一致記号系列が得ることが主な処理となる。従って、最長一致記号系列探索にどのような手法を採用するかにより、符号化全体における性能 (計算時間、記憶容量など) が決定される。最長一致記号系列探索を最も単純に逐次的に実行した場合、最長一致記号系列探索に要する比較回数は最悪で N × M 回となる。従って、比較回数をより削減できるためのデータ構造、探索手法を考案する必要があり、実用化において重要な問題となる。

LZ77 符号のような辞書を用いた符号化法は、記号列の確率的な性質を全く利用せずにモデル化を行なっているように見られる。しかし実際には、LZ77 符号では、辞書に登録された記号列が全て等確率で生起するという情報源モデルを背景に仮定し、それぞれの記号列を等長の符号語に符号化していると考えることができる。

2.3　LZ77 符号 ~ LZ77 復号化 ~

2.3.1　LZ77 復号化アルゴリズム

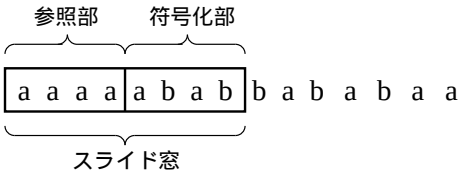
LZ77 復号化は、LZ77 符号化の逆過程を実行するものとなる。以下に復号化アルゴリズムを示す。

(1) 初期化

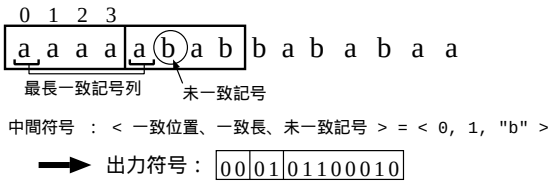
スライド窓 (参照部バッファ + 符号化部バッファ) として、参照部バッファ長 N、符号化部バッファ長 M とする。次に参照部バッファ内を a_0で初期化 (符号化・復号化共の参照部バッファは同じ記号で初期化)。また、現在復号化を行っている記号系列の位置を示すポインタ i = 1 とする。

(2) 記号系列の復元

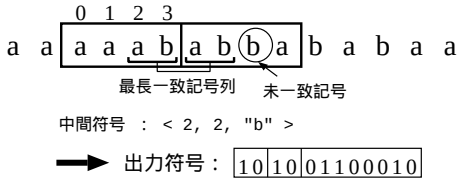
符号語を読み込み、参照部バッファでの一致記号系列の先頭位置 P、一致長 L、符号化部バッファでそれに続く未一致記号 W を決定する。次に P、L により決定された記号系列 S[P, L] を符号化部バッファの先頭から始まる位置にコピーし、それに続



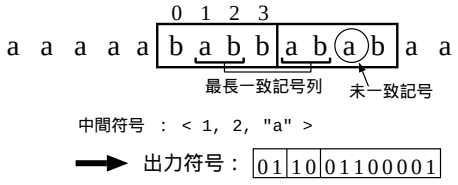
(a) スライド窓の初期化



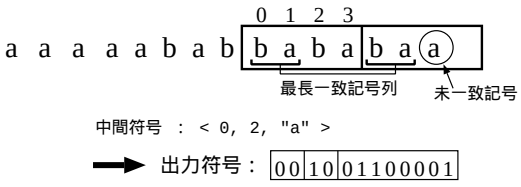
(b) 記号列 "ab" の符号化



(c) 記号列 "abb" の符号化



(d) 記号列 "aba" の符号化



(e) 記号列 "baa" の符号化

図 2.3: LZ77 符号の符号化過程

いて W を置く。また、ここで求められた S[P, L] + W が復元記号系列として出力されることになる。

(3) スライド窓の更新

スライド窓内記号系列を L + 1 だけずらす。また、復号化を行っている記号系列を示すポインタ i を

$$i \leftarrow i + L + 1$$

とした後、(2) へ戻る。全ての符号語が読み込まれ復号化が終了するまでこの作業が継続される。

次に、LZ77 符号化での具体例で符号化されたものを実際に復号化する過程を、上記復号化アルゴリズムに沿って示す。この場合も N = 4、M = 4 である (図 2.4)。

(1) 初期化

初期化により、参照部バッファ内は全て "a" で初期化されている。また i = 1 とする。

(2) 復号化

符号化で求められた最初の符号 000101100010 から、最長一致記号系列の先頭位置 P = 0、一致長 L = 1、未一致記号 W = "b" が決定される。次に、位置 P = 0 から始まる長さ L = 1 の記号列 "a" 、未一致記号 W = "b" が符号化部バッファの先頭部から始まる位置にコピーされ、またこれらコピーされた記号列 "ab" が復元記号列として出力される。

(3) スライド窓の更新

スライド窓内記号系列を 1 + 1 = 2 だけずらす。また i = 1 + 1 + 1 = 3 として、再び復号化処理へ戻る。

(4) 復号化

次の符号語として 2 番目の符号 101001100010 から、P = 2、L = 2、W = "b" がそれぞれ求められる。次に (2) と同様に P、L、W により得られる記号列 "abb" が参照部バッファの先頭から始まる位置にコピーされ、復元記号として出力される。

(5) スライド窓の更新

スライド窓の位置を 2 + 1 = 3 だけずらし、i = 3 + 2 + 1 = 6 として、再び復号化処理に戻る。

以下、全ての符号語を復号化するまで処理を継続する。

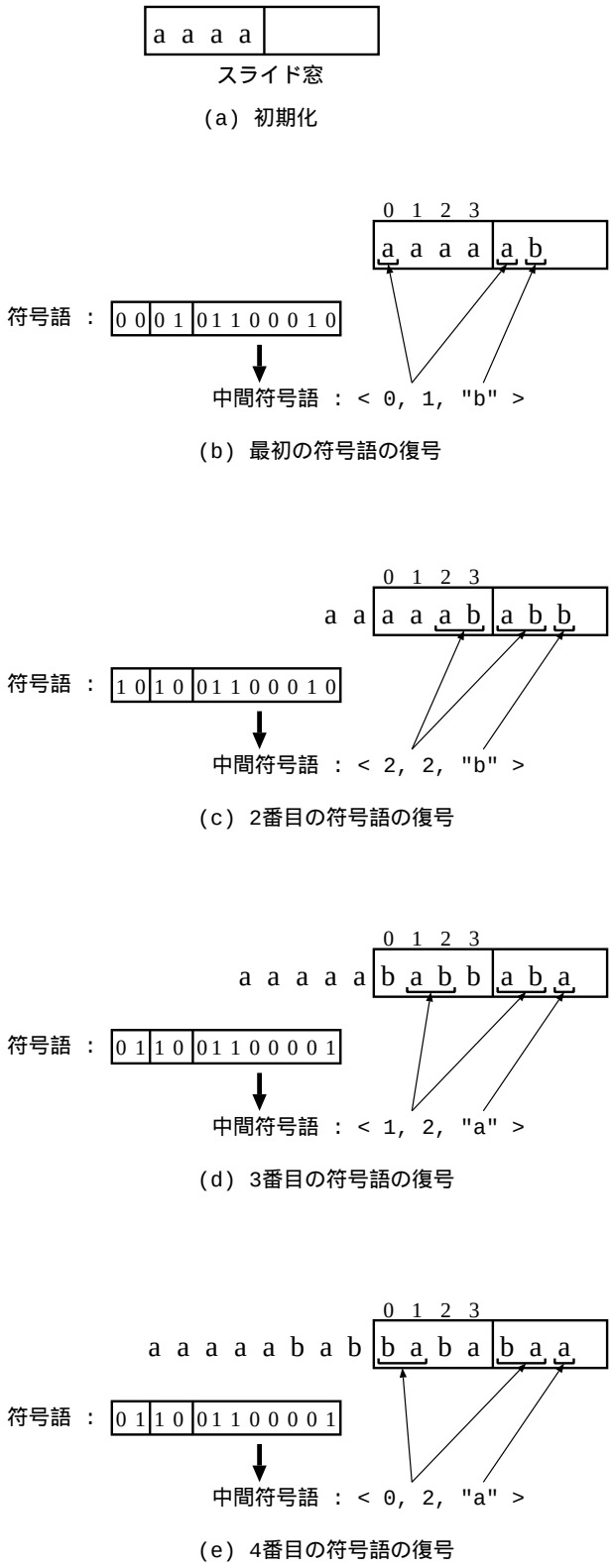


図 2.4: LZ77 符号の復号化過程

2.3.2　LZ77 復号化の特徴

LZ77 復号化では、符号化のときのような最長一致記号系列探索をする必要がなく、主な処理はバッファからの所望記号系列の選択処理となるため、符号化に比べ高速な処理が可能である。以上のことから、LZ77 法は、符号化には時間を要するが、逆に復号化は高速化に実行できるデータ圧縮法である。

2.4　LZ77 符号の改善手法

Ziv と Lempel によるオリジナルの LZ77 符号が提案されて以来、この符号に対し様々な改良法が提案されている。主なものに表 2.1に示すような改良符号がある。

表 2.1: LZ77 符号の改良符号

符号名	提案者	特徴
LZ77	Ziv, Lempel (1977)	ポインタ (一致位置、一致長) と原記号が交互に出現。ポインタは N 記号前までに含まれる部分記号列を指定。
LZR	Rodeh, Pratt, Even (1981)	ポインタと原記号が交互に出現。ポインタは過去の記号に含まれる部分記号列を指定。
LZSS	Storer, Szymanski (1982), Bell(1986)	ポインタと記号をフラグビットで区別。ポインタは N 記号前までに含まれる部分記号列を指定。
LZB	Bell(1987)	LZSS と同様。ポインタの符号化を変更。
LZH	Brent(1987)	LZSS と同様。ポインタの符号化にハフマン符号を利用。
LZBW	Bender, Wolf (1991)	LZSS と同様。一致長として、最長一致記号系列長と 2 番目に長く一致した記号系列長の差を採用
LZIT	伊勢、田中 (1993)	LZSS と同様。スライド窓内の同一部分列による冗長性を除去。

次に、各種改良符号のうちで代表的なものである LZSS、LZB、LZH、LZBW について簡単に述べる。

2.4.1　LZSS 符号

LZSS 符号は、J.A.Storer と T.G.Szymanski が 1982 年に基本的なアルゴリズムを提案し、T.C.Bell が 1986 年に具体的な符号化 / 復号化アルゴリズムとして構成した手法である。

LZSS 符号の特徴は、LZSS 符号での中間符号語形式における冗長性を取り除くことによりデータ圧縮性能を改善している点である。この符号化法は、その後の各種 LZ77 法の改

良符号の基本ともなっていて、現在普及しているデータ圧縮アプリケーションにおいて、LZ77 符号と呼ばれているものの大部分は、実際には LZSS 符号が用いられている場合が多い。

LZ77 符号では、符号語はポインタと記号の組み合せにより構成されている。言い変えると、符号は常にポインタと記号を交互に出力していることになる。この場合、符号には必ず原記号 (圧縮の行なわれていない記号) が存在することになり、この部分が冗長な部分となる。LZSS 符号では、LZ77 符号の中間符号内の冗長性を削減するために、1 ビットのフラグを符号中に設けることにより、ポインタを表す符号と、原記号を表す符号を区別している。すなわち、2 種類の符号形式のうち一方を選択し符号化する。

LZSS 符号では、未一致記号を常に符号に入れることをせず、この未一致記号は符号化部の先頭に置かれることになる。また、符号化部である記号系列を符号化する場合、先頭記号に相当する符号語とポインタに相当する符号語の長さを比較して、常に短い方を採用する手法をとっている。

LZSS 符号の辞書において、辞書に登録されている語は、スライド窓に含まれる長さ T + 1 以上且つ M 以下の記号系列と情報源の 1 記号となる。このことは、LZSS 符号では、あまりに短い記号系列 (1 記号を除く) を辞書に登録しないでおくことで、ポインタを利用して複数の記号をまとめて 1 符号にするという圧縮効果を高めたものとして考えることができる。

2.4.2　LZB 符号

LZSS では、ポインタを表す符号語の長さが一致長によらず一定であった。しかし、一致長がより短いほうが生起しやすいことを考えると、ポインタを表す符号語の長さを可変長にすることで、圧縮効果の向上が見込められる。これに関し、T.C.Bell が LZSS 符号を改善して、ポインタ、記号の表現、記号とポインタを区別するフラグ等に改良を加えたものが LZB 符号である。

LZB 符号では、スライド窓の内容の初期値として、何も書かれていない状態から始められる。入力記号系列の i 番目を符号化する段階で、利用することのできるスライド窓中に蓄えられている記号系列長は、i - 1 または M の大きいほうに一致している。よって i が小さい場合、最長一致記号系列の先頭位置 P としては i - 1 通りしか存在せず、L を $\lceil \log_2(i-1) \rceil$[bit] で表現することができる。つまり、既に符号化された記号数に応じて、最長一致記号系列の先頭位置を表す 2 進数表現のビット長を変化させることができる。

また、ポインタ中の一致長の表現としては、Elisa 符号の符号γ を用いる。これにより、一致長 L を $2\lceil \log_2 L \rceil + 1$ [bit] で表現することができる。LZB では、最長一致記号系列の最大値 M を定める必要がなくなっている。

閾値 T については、実験的には 1 記号が 8[bit] の場合には

$$\mathrm{T} = \lceil (3 + \lceil \log_2(N+M) \rceil)/8 \rceil + 1$$

にすると良いことが知られている。

2.4.3 LZH 符号

LZH 符号は、LZSS 符号で生成される符号における、それぞれのポインタにおける数字の出現頻度を求め、それをハフマン符号を利用して圧縮することで得られる (ハフマン符号の代りに算術符号を用いた手法もある)。

LZH 符号では、始めに LZSS 符号化によって中間符号を決定する。次にこの中間符号に対しハフマン符号を利用するわけだが、このとき、ポインタ中の先頭位置 (N 種類)、一致長 (M 種類)、記号 (記号を表現するために必要なビット数に相当する数 : W) をそれぞれ別個にハフマン符号化しようとすると、N + M + W 個の符号語のテーブルが必要となり、この符号語のテーブルのオーバヘッド が大きくなり、実用性を欠くことになる。そこで LZH 符号では、先頭位置を表す 2 進数表現を上位ビットと下位ビットの二つに分け、二つの数字によって先頭位置を表現することにしている。このようにすることで、$\sqrt{N+M}$ までの数字二つで先頭位置を表現することができる。そして、ポインタを表す数字と記号をあわせたものを情報源記号とみなし、ハフマン符号を構成する。これによって、符号語のテーブルのサイズは、ほぼ $\sqrt{N+M+W}$ となり、それぞれを独立に符号化した場合よりもハフマン符号のために計算量を減少させている。また、このようなハフマン符号化を行うことによって、記号とポインタを区別するためのフラグは必要としなくなる。

なお LZH 符号は、LHA や gzip など広く用いられているデータ圧縮アプリケーションなどにも利用されている。

2.4.4 LZBW 符号

LZBW 符号は P.E.Bender と J.K.Wolf により提案された符号で、LZSS 符号における辞書の冗長性を削除することによって高性能を達成する手法である。この符号の辞書は LZSS と同様のものとなる。しかし、この符号では、スライド 窓の中には同一の記号列が複数回現れることに着目している。このことは辞書の冗長性を意味している。この冗長性を削除するため、最長一致記号系列 (長さ L_1) のほかに、二番目に長く一致した記号系列 (長さ L_2 とする。但し、最長一致記号系列より後に出現し、その記号系列はスライド 窓に完全に含まれるものとする) を求め、これら二つの記号系列の長さの差 $delta = L_1 - L_2$ を求める。delta は、スライド 窓中に頻繁に出現する語ほど小さくなる。そして、LZBW 符号における辞書のポインタとして、最長一致記号系列の先頭位置と delta の組を採用する。符号語中の $delta$ の表現として、LZB 符号のときと同様な整数表現を用いることにより、記号列中によく出現する語に対しては、短い符号を割り当てることが可能となる。

復号化については、始めに最長一致記号系列の先頭位置を求め、この位置より後ろのスライド 窓中で、この記号系列に最長一致する記号系列の長さ (L_2) を求め、この長さに delta を加えたものが求める最長一致記号系列の長さ (L_1) となる。

2.5　LZ77 符号の統計的特徴

2.5.1　LZ77 符号化での最長一致記号系列探索

LZ77 符号化において大部分の計算は、参照部バッファ(辞書バッファ) と符号化部バッファ(次に符号化される記号列) との間の最長一致記号系列探索に費やされる。よって、最長一致記号系列探索に要する計算量、すなわち記号比較回数をいかに小さくできるかが、高速な符号化における大きな鍵となる。

ここで参照部バッファ長 N、符号化部バッファ長 M とする。ソフトウェアによる処理では、逐次的な記号系列探索を行わなければならない。この場合最も基本的な手法では一符号あたり O(NM) の計算量を要する。また他の記号列照合アルゴリズム応用、または木などを用いた探索法などを用いることで、一符号あたり O(N) 程度の計算量での一致長判定が実行できる。しかしその分処理が繁雑になり、十分な性能 (速度、メモリ容量) での一致長判定処理が困難となる。

現在、一致記号系列探索がハードウェアとして実装されている分野の代表的な一つとして、テキスト・データベースにおける文字列検索の分野がある。テキスト・データベースの分野では、一致記号系列探索 (文字列検索) 用ハードウェアの代表的なアーキテクチャの一つとして、一次元シストリックアレイ (セルラ・アレイ) により一致記号系列を探索する手法がある [10] [11]。一次元シストリックアレイによるアーキテクチャ[14] [15] は、

- 同じ計算セルを複数に接続し、パイプライン的に一致記号系列を探索する。
- 一定方向の記号系列の流れ (セル間でのデータ依存がない) により探索される。

という特徴を持つ。一次元シストリックアレイによる実現は、同じ計算セルを利用できる、データの流れが一定方向であるという理由により、VLSI 化がより容易なアーキテクチャである。以上の理由により、LZ77 符号化アーキテクチャの代表的な方式として、最長一致記号系列探索に一次元シストリックアレイを利用したアーキテクチャが提案されている [1] [4]。

2.5.2　生成符号の統計的特徴を利用した計算量の削減

現在提案されている LZ77 符号化では、ソフトウェア・ハードウェアによる手法ともに、一符号あたり O(N) の計算量を実現しているものが一般的である。ここで、実際に求められる最長一致記号系列長について考える。LZ77 符号化において必要となるのは、参照部バッファ内のある位置からの記号列と、符号化部バッファの先頭位置からの記号列との、最長一致記号系列を探索することである。次に、実際には最長一致記号系列探索が $L + 1$ 回の計算 (比較演算) 回数で実行可能であること、及びその並列アルゴリズムを提案する。

一致記号系列探索過程では、$L + 1$ 回目の一致判定で不一致が生じることにより、結果的に長さ L の一致記号系列が求められる。仮に、求められる最長一致記号系列長を L とすると、この記号系列のみに注目した場合、少なくとも $L + 1$ 回の記号比較処理を実行す

れば所望の最長一致記号系列を探索することができる。しかも最長一致記号系列は、一回の符号化により得られる参照部バッファの全ての位置の一致記号系列の中では最長である。この性質から、本論文で提案する最長一致記号系列探索の考え方を図 2.5に示す。

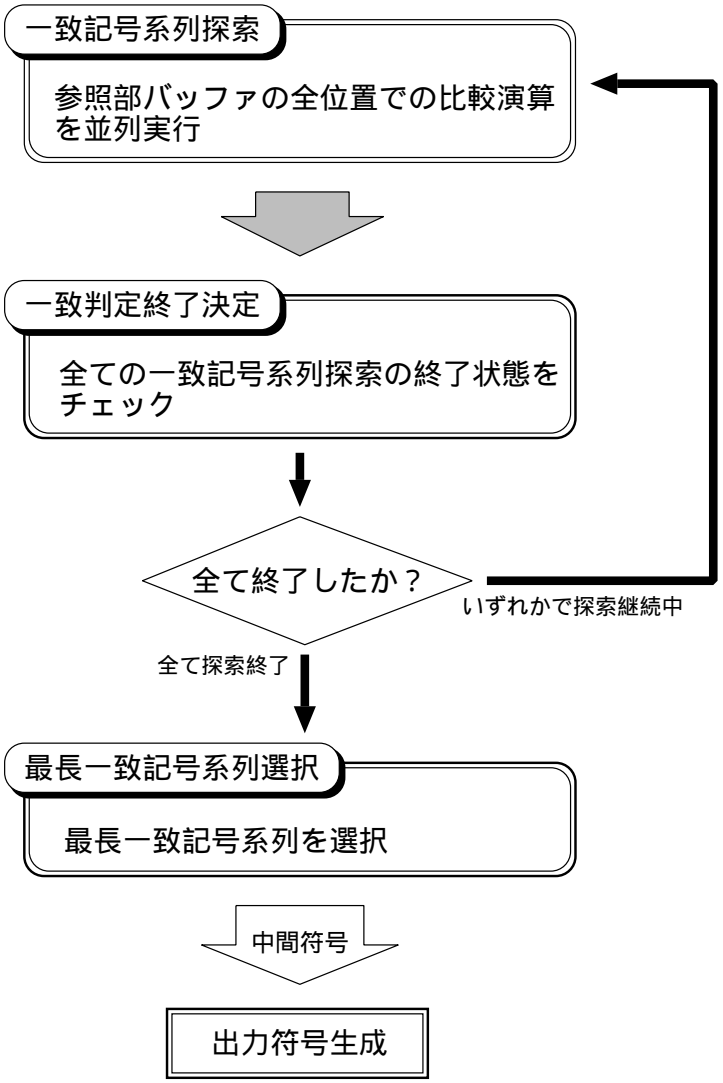


図 2.5: 最長一致記号系列探索並列アルゴリズム

図 2.5の提案並列アルゴリズムにより、参照部バッファの全ての位置の一致記号系列は L + 1 回の比較演算により探索できる (以下、一致記号系列探索部と呼ぶ)。その後、その中から適時最長のものを選択する (以下、最長一致記号系列選択部と呼ぶ) ことで、所望の最長一致記号系列を探索できる。また提案並列アルゴリズムでは、

- 最長一致記号系列選択部は、参照部バッファの全位置の一致記号系列が探索される毎に実行される。

- 最長一致記号系列選択部は、パイプライン的に並列処理可能である。

ということから、一致記号系列探索部とは独立・並列動作する機能として実装することが可能である。また最長一致記号系列選択部は、現実にはパイプライン的な動作により、一符号あたりの計算量は O(1) にすることが可能である。従って、上記の手法により最長一致記号系列探索を実現することで、計算量は一致記号系列探索部に依存し、その比較演算は L + 1 回で実現することができる。

以上のプロセスにより、最長一致記号系列探索を一符号あたり L + 1 回の比較演算で実行可能なアーキテクチャを構築できる。ここで、L、M(符号化部バッファ長。最長一致記号系列長の最大長を定義する)、N には次の不等式が成立する。

$$\mathbf{L} \leqq \mathbf{M} < \mathbf{N} \tag{2.1}$$

従って、L は N に比べ小さいものである。この上で、L が N に比べて十分に小さいものであれば、一符号あたり O(N) で実現されるシストリックアレイを用いたアーキテクチャに比べ、最長一致記号系列探索に必要な計算量を大幅に削減できる。

そこで次に、実際に LZ77 符号化で得られる最長一致長の統計的特徴 (生成符号の統計的特徴) について調査した。以後、一記号を 1[byte] とし、参照部バッファ長 N[byte]、符号化部バッファ長 M[byte]、最長一致記号系列長 L[byte] とする。LZ77 符号は可逆圧縮法の代表的な一つであり、テキストデータや実行形式のファイルなど、圧縮前と復元後のデータ形式が全く同一でなければならない場合に用いられるデータ圧縮手法である。そこで本研究では、統計的特徴の調査には、複数のテキスト形式のデータファイルをサンプルファイルとして使用した。サンプルファイルは表 2.2の 12 種類のデータファイルである。

2.5.3　1 データファイルあたりの最長一致記号系列長の頻度 ~ 一符号あたりに必要な比較演算回数 ~

始めに、一つの符号がどの程度の長さの L を持っているか、すなわち一つの符号を生成するのに最低どの程度の比較演算処理が必要となるのかについて調査した。ここでは、一つのデータファイルを符号化した場合について、一符号の有する L の頻度として求めた。次に各データファイルでの頻度分布を示す。

ここで示している各図 (図 2.6、2.7、2.8、2.9、2.10、2.11、2.12、2.13、2.14、2.15、2.16、2.17) は、N=1024、M=128 の条件での頻度分布である。頻度分布の状態は、N、M のパラメータ値をを増加させるつれて、分布が若干右側へ広がっていくことが確認された。しかし、ほとんど変化しないという結果であると言ってよい程度の変化であった。従って頻度分布については、各バッファ長によらずほぼ同様の分布を示しているものと判断される。また、どのデータファイルの場合での頻度分布についても、ほぼ同様の分布を示していることが確認された。

頻度分布から、生成される符号の有する一致長 L の大部分は 0 ~ 20 程度の間に集中 (0 ~ 10 で全体の約 94 %、0 ~ 20 で約 98 %) しており、L が N に相当するような値になる

表 2.2: サンプルデータ

ファイル名	ファイルサイズ [byte]	フォーマット
2ByteText.txt	71970	全角文字テキスト
C-code1.txt	3799	C ソースコード
C-code2.txt	198506	同上
PSFile1.txt	343299	PS ソースファイル
PSFile2.txt	571290	同上
Retrieve1.txt	27560	inspec での検索結果
TextFile1.txt	224796	ascii code テキスト
TextFile2.txt	236036	同上
TextFile3.txt	217284	同上
TextFile4.txt	123608	同上
TextFile5.txt	129941	同上
Unix-man1.txt	28939	UNIX manual page

ことはほとんどないことが確認された。この結果は、符号を生成するときに最低必要となる比較演算回数が、ほとんどの場合で 1 ~ 20 程度であることを示すものであり、実用上の N (1024 ~) に比べ十分に小さいものである。同時に、アーキテクチャの実現にあたっての M の値としても、M = 32 程度で十分であると判断できる。

2.5.4 一符号あたりの平均比較回数

得られた頻度分布から、符号化にあたっての一符号あたりの平均比較回数を求めることができる。各データファイルにおける一符号あたりの平均比較回数を示す。

表 2.3は、最長一致記号系列長の頻度の場合と同様に、N=1024、M=128 の条件での一符号あたりの平均比較回数である。平均比較回数は各データファイルにおいても、N が大きくなるに従って増加していく。しかし調査の結果では N=4096[byte] 程度 (一般的によく用いられる値) でも最大 8 程度の値であった。図 2.18、2.19、2.20、2.21、2.22、2.23、2.24、2.25、2.26、2.27、2.28、2.29に、各サンプルデータファイルでの平均比較回数の変化のグラフを示す。

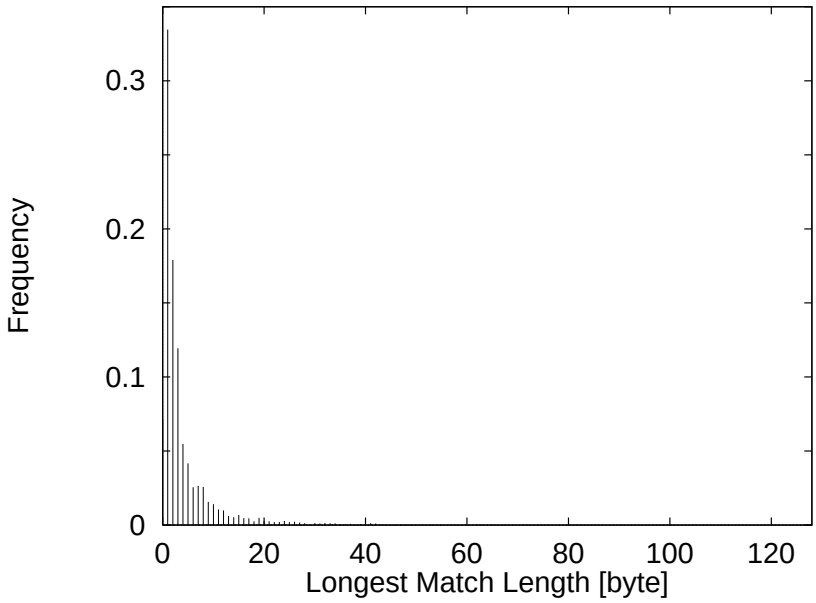


図 2.6: 2ByteText.txt での頻度分布

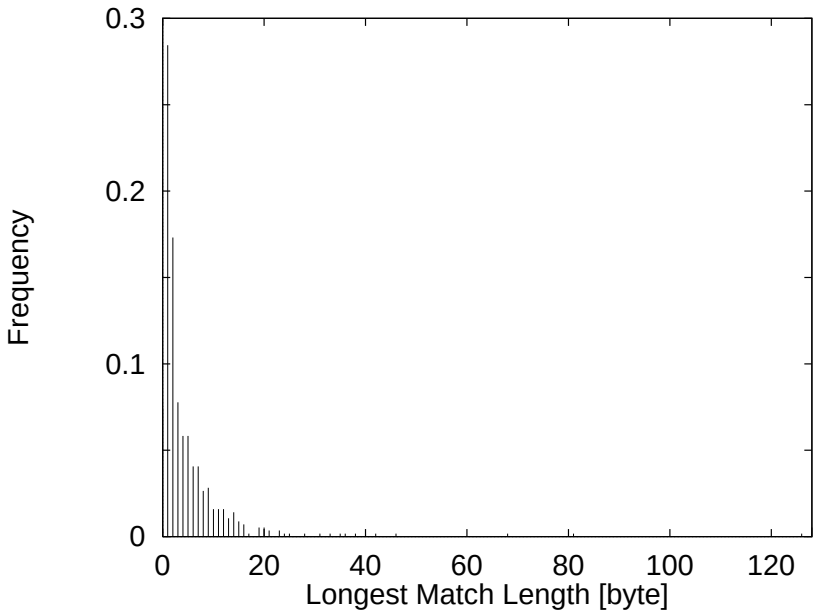


図 2.7: C-code1.txt での頻度分布

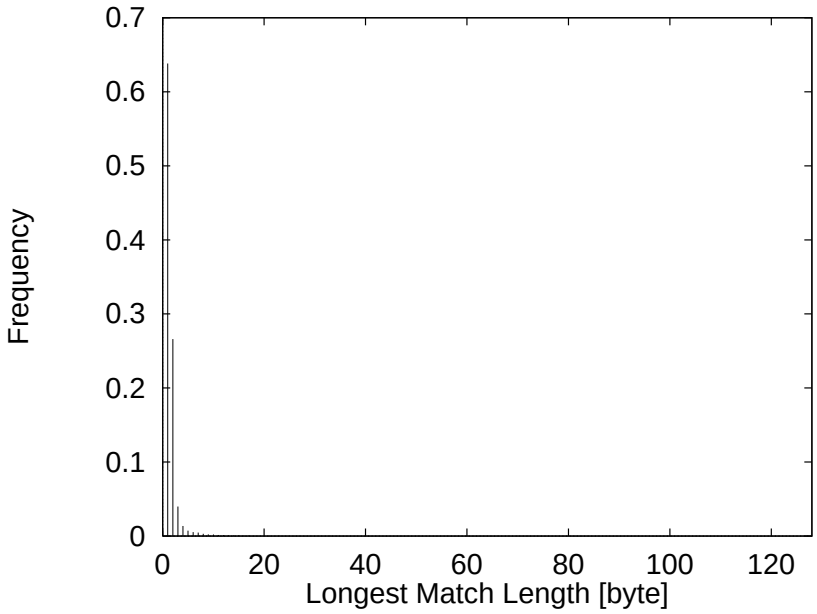


図 2.8: C-code2.txt での頻度分布

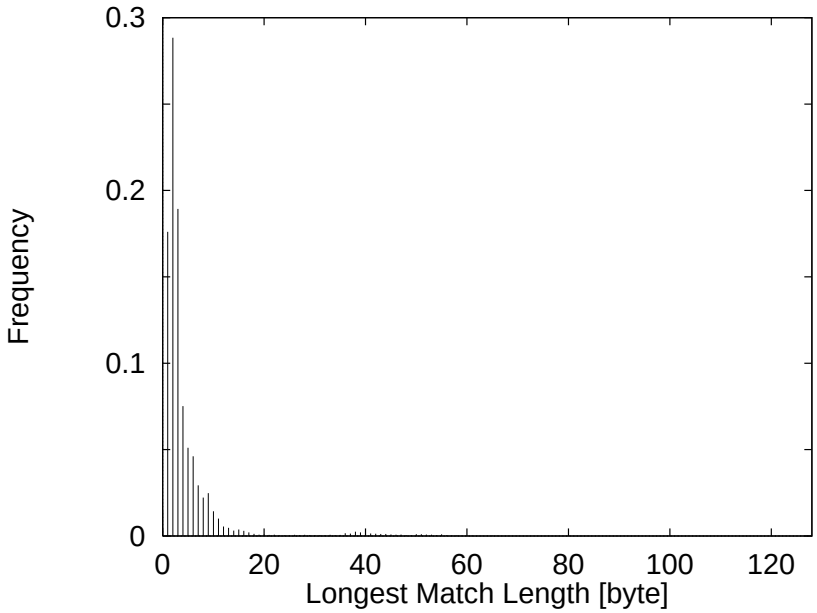


図 2.9: PSFile1.txt での頻度分布

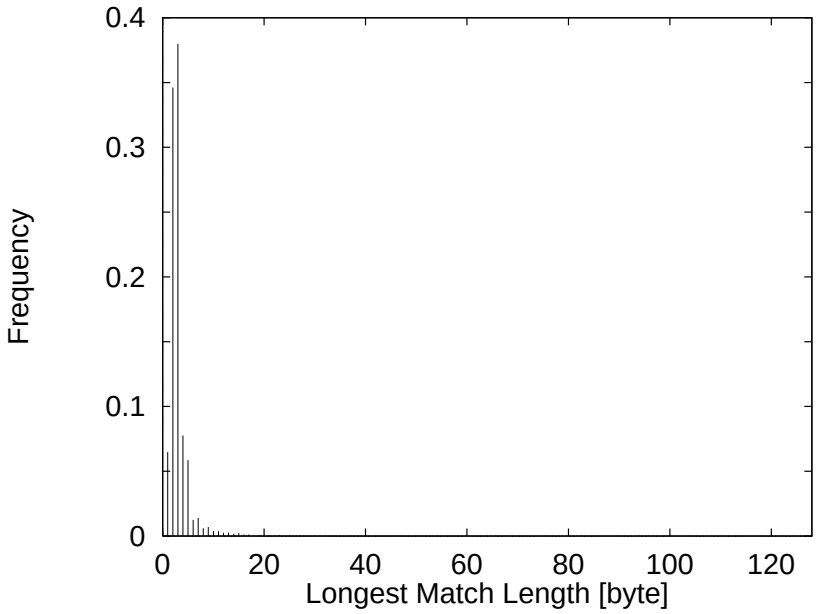


図 2.10: PSFile2.txt での頻度分布

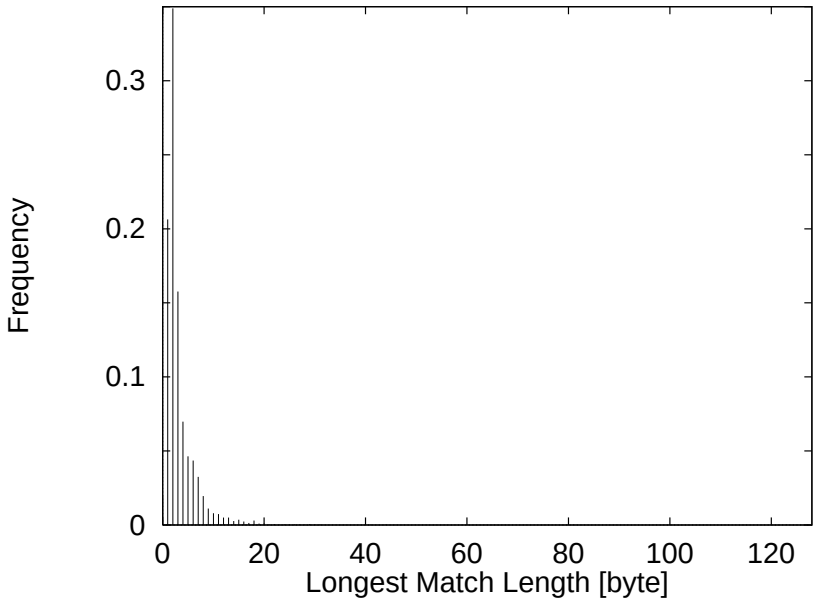


図 2.11: Retrieve1.txt での頻度分布

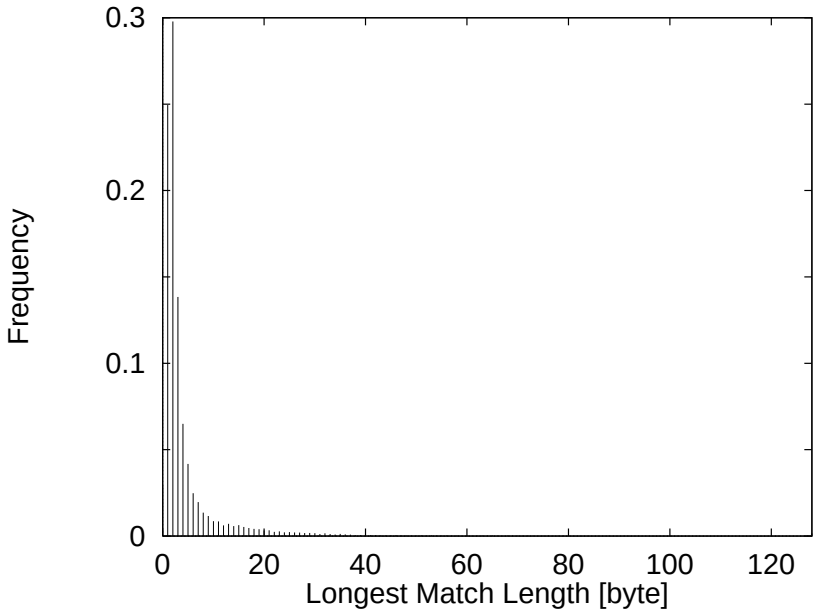


図 2.12: TextFile1.txt での頻度分布

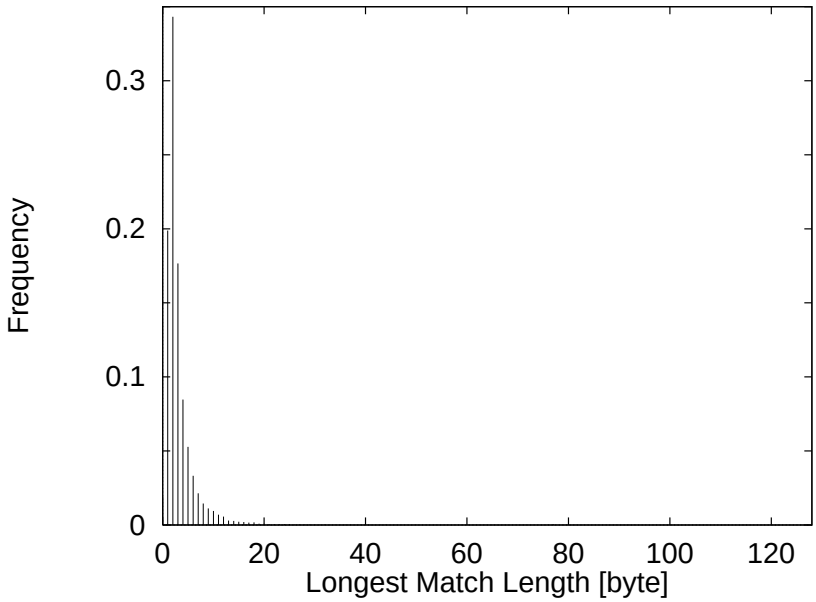


図 2.13: TextFile2.txt での頻度分布

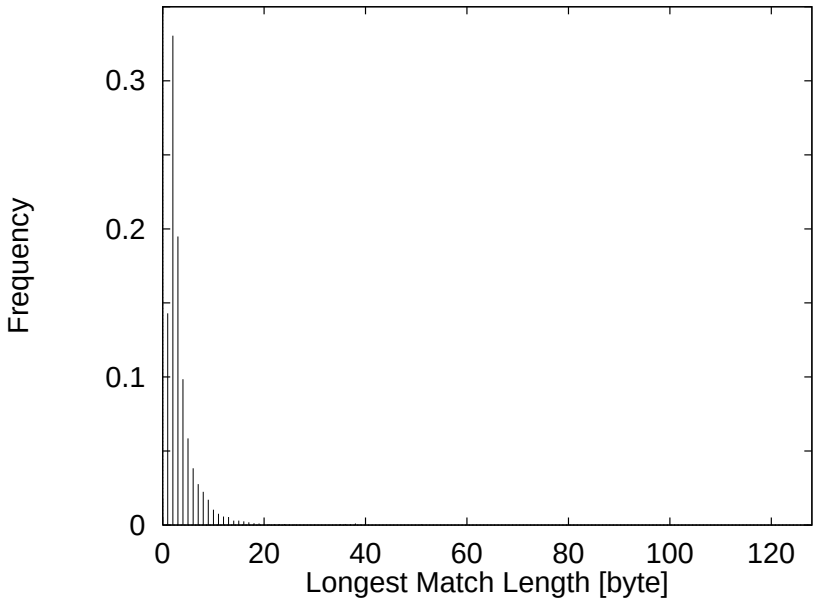


図 2.14: TextFile3.txt での頻度分布

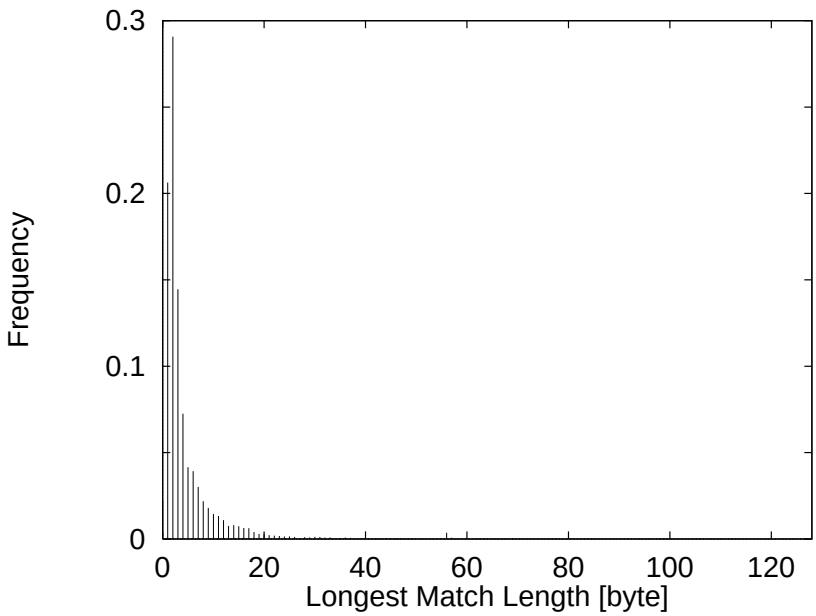


図 2.15: TextFile4.txt での頻度分布

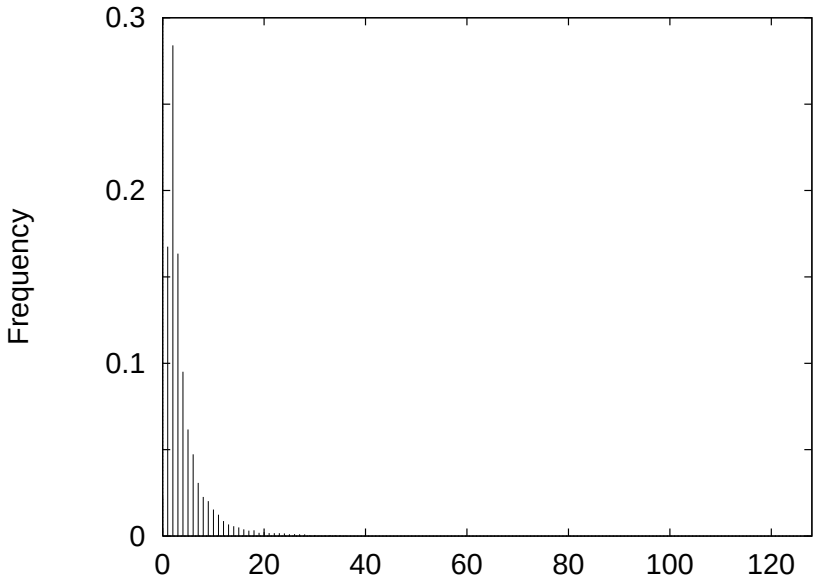


図 2.16: TextFile5.txt での頻度分布

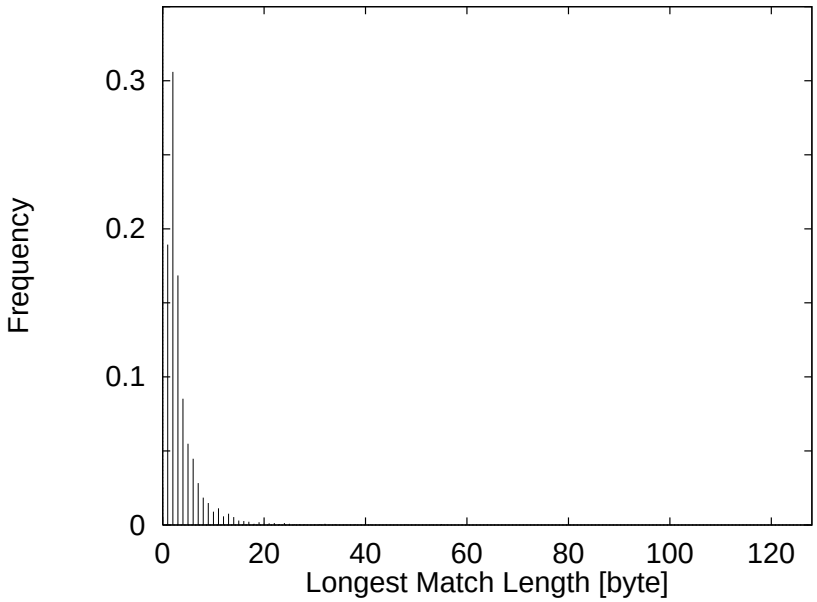


図 2.17: Unix-man1.txt での頻度分布

表 2.3: 各データファイルにおける平均比較回数

ファイル名	平均比較回数
2ByteText.txt	6.25
C-code1.txt	7.71
C-code2.txt	3.89
PSFile1.txt	6.76
PSFile2.txt	5.50
Retrieve1.txt	5.30
TextFile1.txt	6.91
TextFile2.txt	5.31
TextFile3.txt	5.69
TextFile4.txt	6.70
TextFile5.txt	6.13
Unix-man1.txt	5.81

(全体での平均比較回数 ： 6.04)

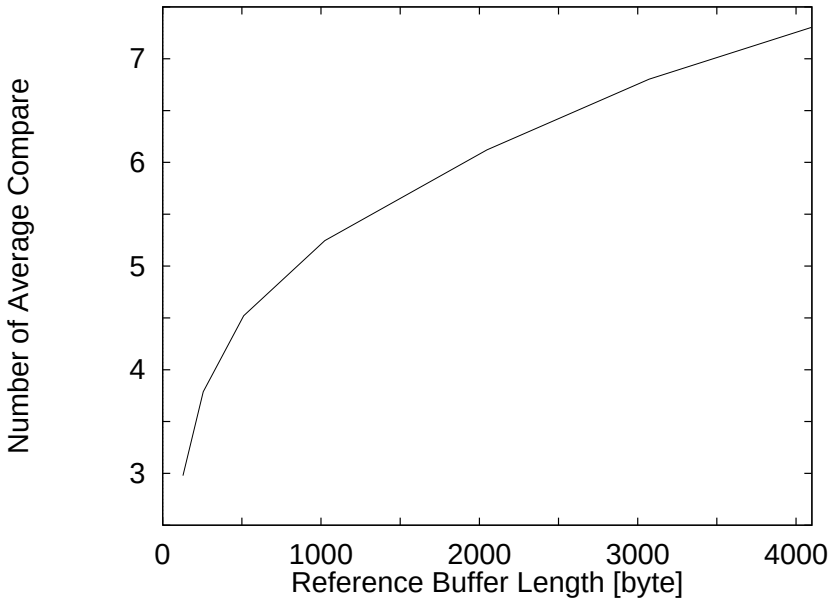


図 2.18: 2ByteText.txt での平均比較回数

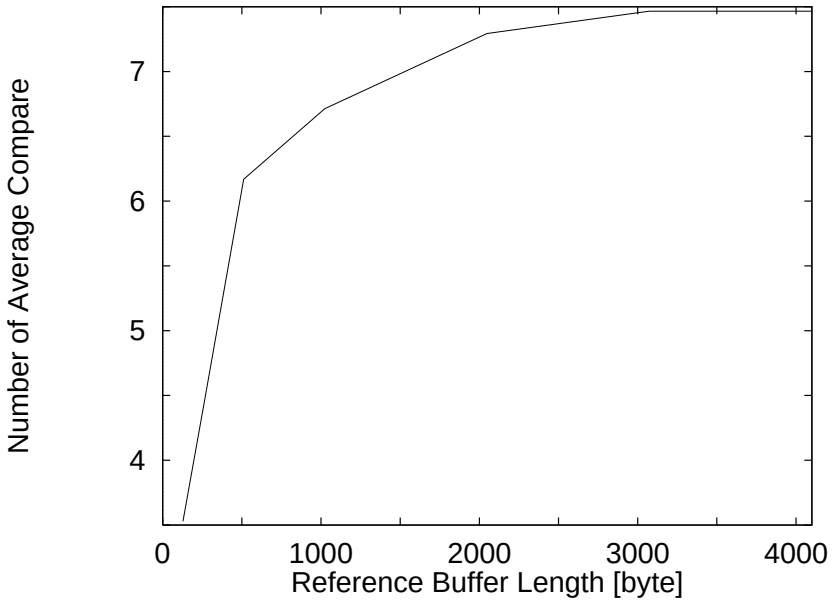


図 2.19: C-code1.txt での平均比較回数

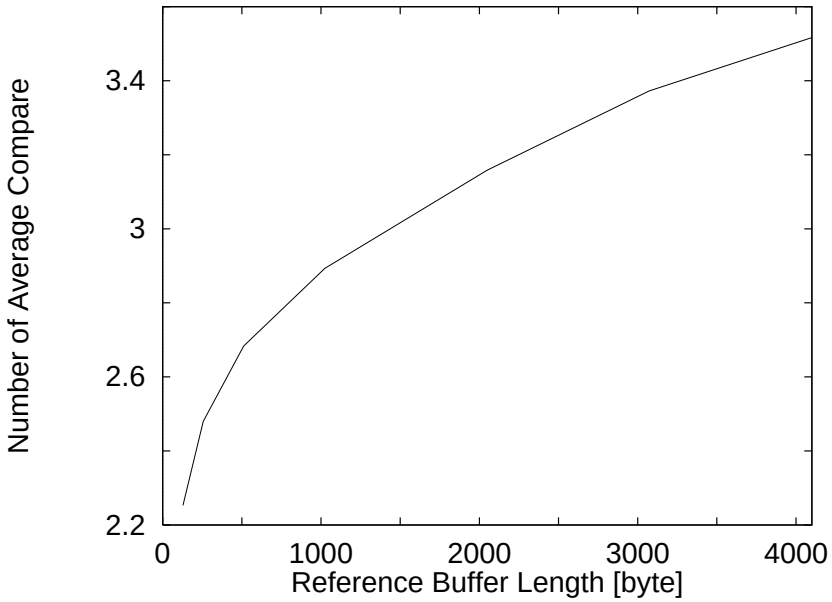


図 2.20: C-code2.txt での平均比較回数

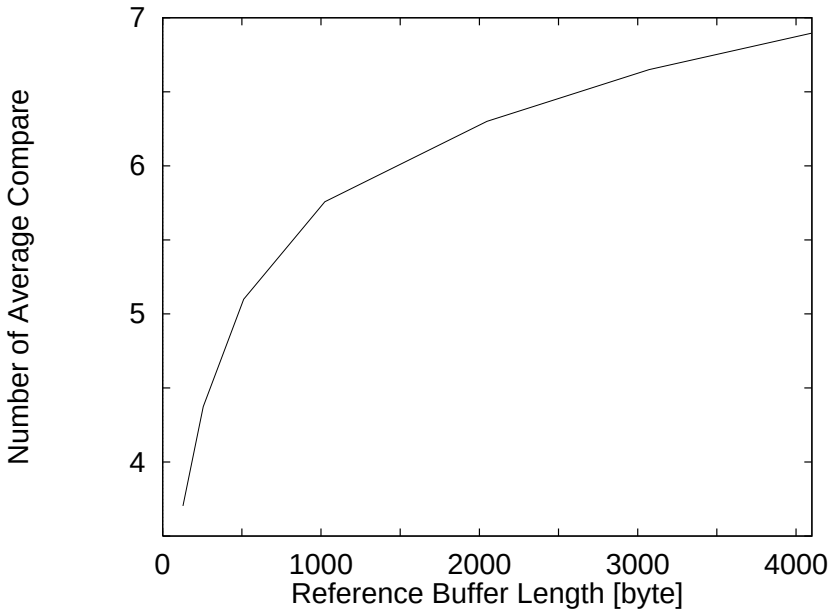


図 2.21: PSFile1.txt での平均比較回数

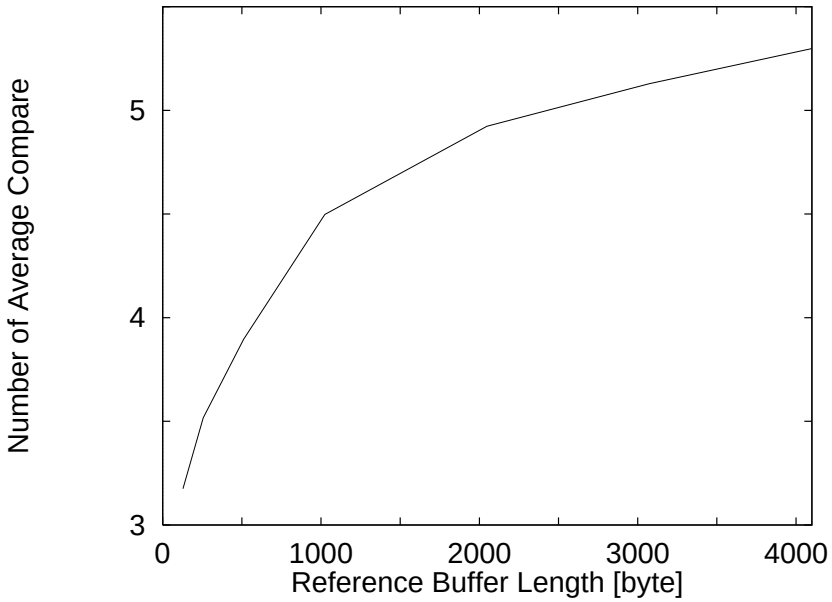


図 2.22: PSFile2.txt での平均比較回数

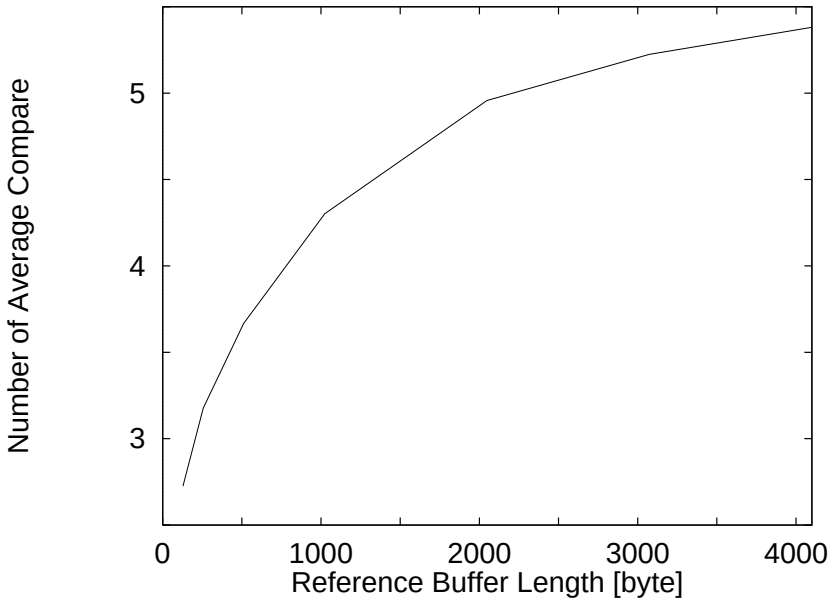


図 2.23: Retrieve1.txt での平均比較回数

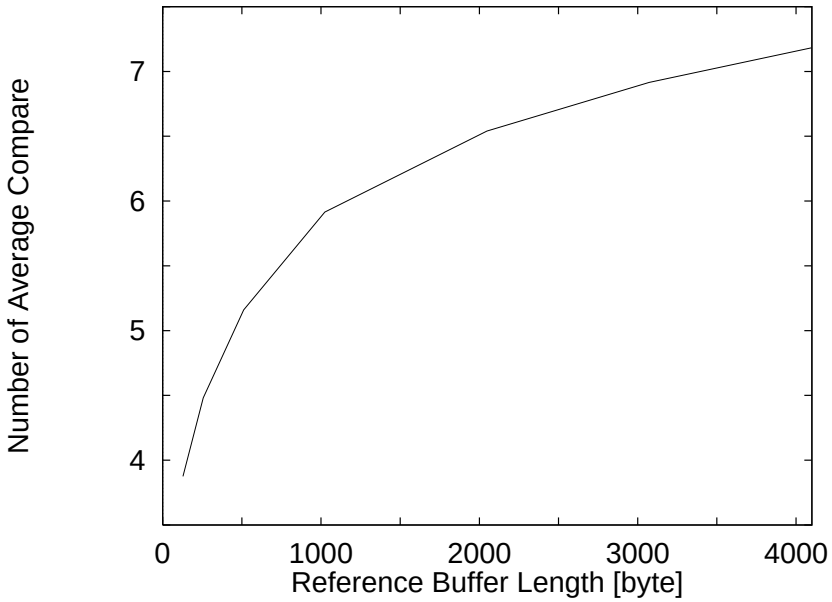


図 2.24: TextFile1.txt での平均比較回数

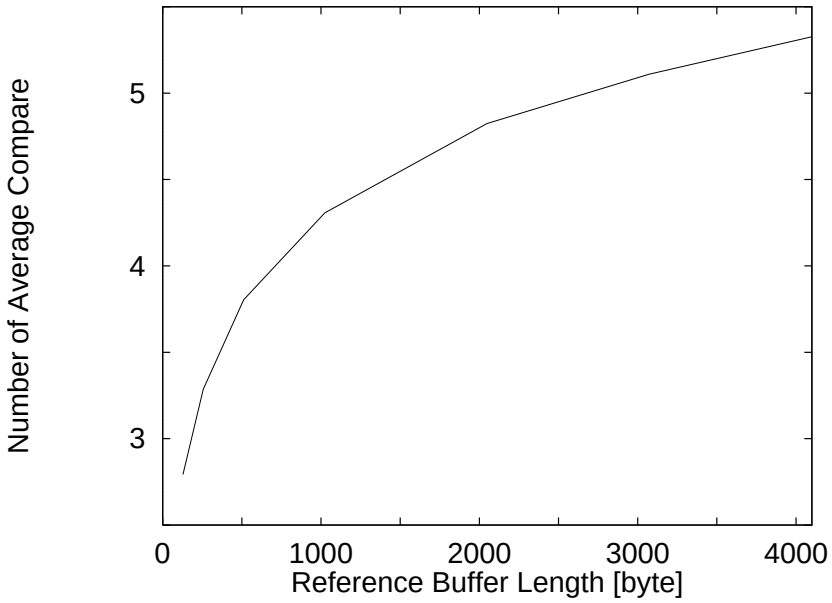


図 2.25: TextFile2.txt での平均比較回数

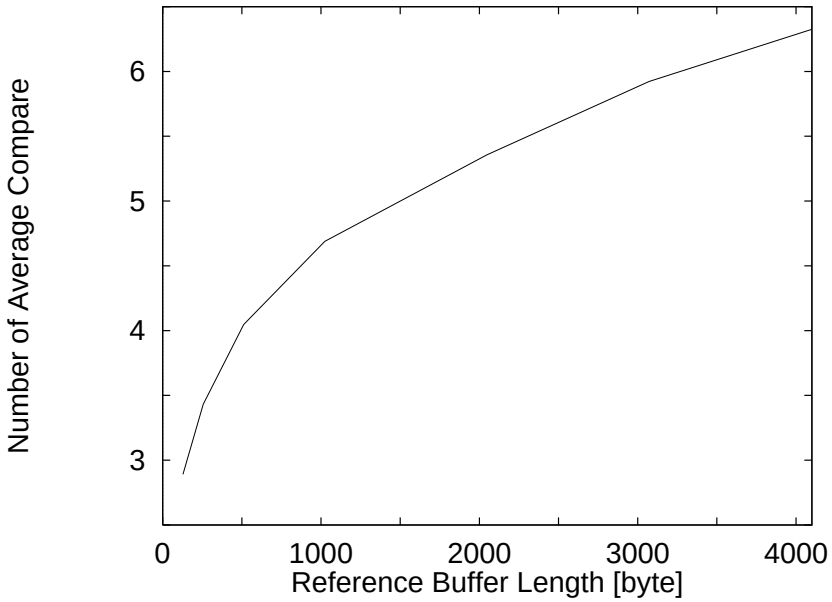


図 2.26: TextFile3.txt での平均比較回数

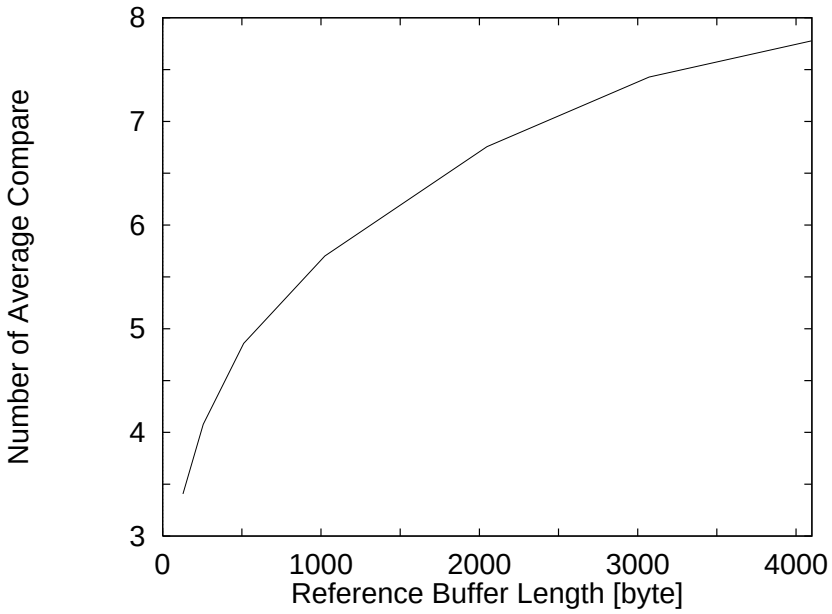


図 2.27: TextFile4.txt での平均比較回数

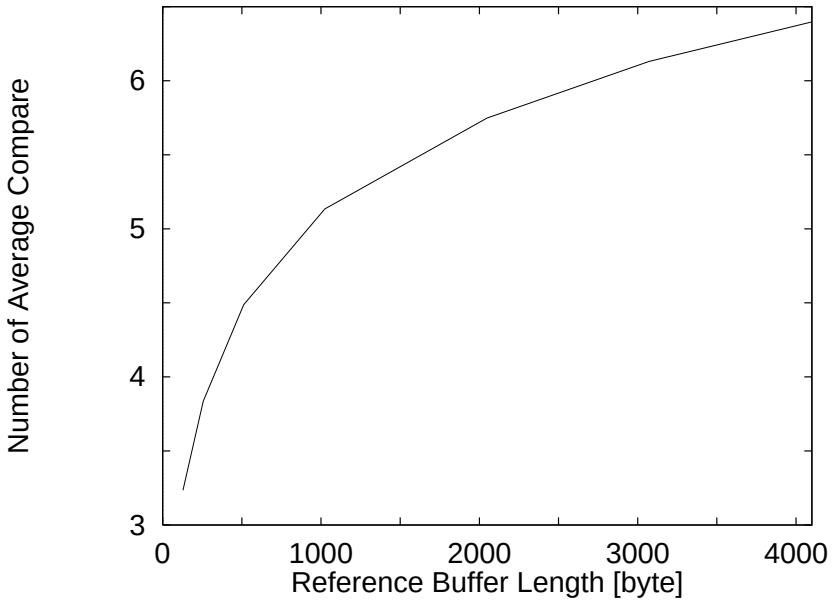


図 2.28: TextFile5.txt での平均比較回数

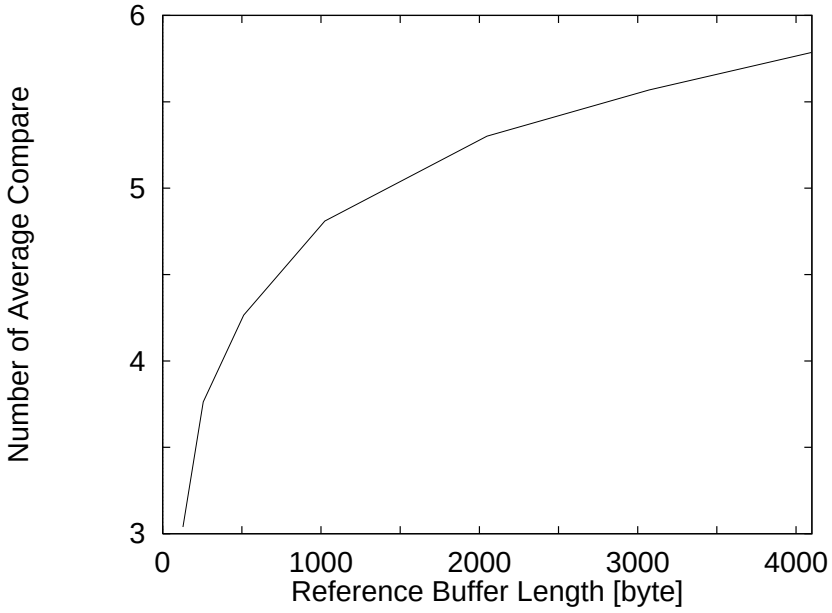


図 2.29: Unix-man1.txt での平均比較回数

2.5.5 符号の統計的特徴を利用した場合の計算量

LZ77 法での符号の統計的統計的特徴から、一つの符号を得るのに最低必要な比較演算回数は、N に比べ十分小さいものであることが確認された。この特徴から、一符号あたり L + 1 回の比較演算回数による符号化が、符号化における計算量の削減にどの位有効であるのかについて、次の例を示す。

- 参照部バッファ長 ： N [byte]
- 全生成符号数 ： X
- 一符号あたりの平均比較回数 ： A

とすると、一つのデータファイルを符号化する場合に必要な比較演算回数 (== 計算量) は次のようになる。

- ○ 既存アーキテクチャ(シストリックアレイによる手法)： $X(2N-1)$
- ○ 生成符号の統計的特徴を利用した手法： XA

次に、実際のデータファイルにより値から求められる総比較演算回数を示す。

(例)**N=128** の場合

- データファイル： 198506[byte] の C source file

- 得られた X ： 88106
- 得られた A ： 3.253

○ 既存アーキテクチャ：

$$88106 \times (2 \times 128 - 1) = 22467030$$

○ 統計的特徴利用アーキテクチャ：

$$88106 \times 3.253 = 286609$$

上記のパラメータ条件及びデータファイルでの結果では、既存アーキテクチャに比べ、計算量 (比較演算回数) を $1/80$ 程度 $(1.27[\%])$ まで削減することが可能であることが確認された。以上の例からもわかるように、最長一致記号系列探索における、一符号あたりの比較演算を $\mathrm{L} + 1$ 回にすることにより、LZ77 符号化の計算量の大幅に削減できる。よって、計算量の削減により符号化時間が削減され、高速な符号化が実現される。

第 3 章

高速 LZ77 符号化・復号化並列処理アーキテクチャ - PAHL

3.1 高速 LZ77 符号化並列処理アーキテクチャ - PAHL-C

3.1.1 提案アーキテクチャに必要な機能

第 2 章では、最長一致記号系列探索 (最長一致記号系列長：L) を一符号あたり $L + 1$ 回の比較演算回数で実現する並列アルゴリズムを提案し、LZ77 符号における実際の生成符号の統計的特徴を調査した。その結果、一つの符号を生成するのに要する計算量 (最長一致記号系列探索における記号一致比較演算) を $L + 1$ 回にすることが可能であり、符号化に要する計算量を大幅に削減できることを示した。

LZ77 符号化における最長一致長探索では、次に符号化される記号系列 (符号化部バッファの先頭位置から始まる記号系列) と、参照部バッファ (辞書バッファ) の各位置から始まる記号系列との一致記号系列を探索し、その中から一致長の最大の記号系列を求めるという処理を行う。その処理により得られた最長一致記号系列の、参照部バッファ上での位置、一致長、及び符号化部バッファ上でのその最長一致記号系列に続く未一致記号を、適当な 2 進数表現したものを符号として出力する。

一符号あたり $L + 1$ 回の比較演算での符号化 (最長一致記号系列探索) を実行するための具体的な並列アルゴリズムは次の通りである。

(1) 参照部バッファの全位置の一致記号系列探索を並列に実行。

- 参照部バッファの全位置での比較演算を並列実行。
- 一回の比較演算の度に、各位置で一致記号系列探索が完了したか否かをチェック。全位置での探索が完了するまで比較演算が継続される。

(2) 全位置での一致記号系列探索が完了した時点で、一符号あたりの一致記号系列探索を終了。

(3) 得られた全位置での一致記号系列の中から最長記号系列を選択→ 最長一致記号系列。

一致記号系列探索部 ((1)(2)) により、参照部バッファの全位置での一致記号系列探索が終了するのは、L + 1 回の比較演算を行なった時点となる。すなわち一符号あたりの一致記号系列探索部に要する比較演算は L + 1 回である。なお、全位置での一致記号系列探索が終了した時点で、生成符号に必要となる情報 (データ) は以下のものとなる。

- 参照部バッファの全位置での一致記号系列探索終了時のカウント値 C。この値から最長一致記号系列長が得られる (正確には終了の前時点でのカウント値)。
- 参照部バッファの全位置での一致記号系列探索終了時の前時点において、それぞれの位置で一致記号系列探索が継続中であったか否かの状態を示す値 S_n(n = 0 ~ N-1)。それぞれ 0 または 1 のフラグとして持っておけばよい。
- 参照部バッファの各位置を示す値 P。
- 参照部バッファの全位置での一致記号系列探索終了時に符号化部バッファの先頭位置に存在する記号 W。この記号が未一致記号となる。

最長一致記号系列選択部 ((3)) では、参照部バッファの各位置から始まる一致記号系列の中から、最長一致記号系列の参照部バッファ上の位置を求めることになるが、この処理は単に (1)(2) により得られた S_0 ~ S_{N-1} を見ることにより、所望のものを選択することができる。

最長一致記号系列選択部は、一致記号系列探索部により得られた値を必要とするが、一致記号系列探索部は最長一致記号系列選択部での結果に依存しない。つまり、一つの符号を生成するために (1)(2) の処理により得られた値を (3) の処理へ転送した後は、その値による (3) の処理が終了しているか否かにかかわらず、次の符号生成のための (1)(2) の処理を実行することができる。すなわち (1)(2) と (3) の処理は、それぞれ独立に並列処理することができる。このことから、LZ77 符号化の計算量は (1)(2) か (3) のいずれかの処理の計算量に依存する。(3) の処理は木構造のシストリックアレイにより、パイプライン的に容易に実現でき、その場合の木の高さは $O(\lceil \log_2 N \rceil)$ となる。しかし、(3) は (1)(2) の処理に依存せず、逐次データ入力可能な部分であり、一符号あたりでは O(1) となる。従って、実際の符号化の計算量は L の大きさに依存することになる。以上のアルゴリズム ((1)、(2)、(3) の処理) を実現することにより、L + 1 回の比較演算での (O(L) での) 符号化を実現する高速 LZ77 符号化並列処理アーキテクチャについて次に述べる。

3.1.2　高速 LZ77 符号化並列処理アーキテクチャ - PAHL-C の基本構成

一符号あたり L + 1 回の比較演算での符号化のために前節で提案した最長一致記号系列探索並列アルゴリズムを実現した、高速 LZ77 符号化並列処理アーキテクチャ- PAHL-C

(Parallel Architecture for High-Speed LZ77 Data Coding : PAHL-C) の基本構成、動作アルゴリズムについて述べる。その基本構成を図 3.1に示す。

PAHL-C の動作アルゴリズムは以下のようになる。

(1) 参照部バッファ(辞書バッファ)、符号化部バッファ(の先頭位置用バッファ)、最長一致記号系列選択部 (cell2 部)、一致長、未一致記号遅延用バッファの初期化。

(2) 一致記号系列探索部 (cell1 部)、カウンタの初期化。

- cell1 - 一致長決定信号を 0 にセット
- counter - カウント値を 0 にセット

(3) 全 cell1 での一致記号系列探索

(a) 各 cell1 での (一致記号系列探索における) 比較演算の並列実行 →一致長決定信号
- 不一致が生じた時点で一致長決定信号を 1 にセット
- 一致長決定信号が 1 の場合、次の初期化までその状態を保存

(b) 各 cell1 からの一致長決定信号による一致長探索継続の判断
all-match-cell - 全 cell1 からの一致長決定信号の AND →一致判定継続信号

(c) 各 cell1 の前回の一致長決定信号の NOT と現在の一致判定継続信号との AND、及び cell1 の番号を、各々cell2 部へ出力 → (各 cell1 からの) 最長一致信号

(d) カウンタの更新

(e) 参照部バッファ、符号化部バッファ内記号の更新

(a) ~ (c) までと (d)、(e) は並列に実行される。また (a) ~ (e) は一致判定継続終了の判断が出るまで継続される。

(4) 得られた各 cell1 からのデータから所望のデータを選択。

(最長一致記号系列 (位置) 選択部 ： cell2 部)

- (3) での処理により得られた (一致判定終了時点の前時点での) 一致長決定信号の NOT と現在の一致判定継続信号との AND が、各 cell1 の最長一致信号となる。最長一致信号により、最長一致記号系列の得られた cell1 の番号を選択。この番号が一致位置となる。
  (i) 各 cell2 には、前段から二つの cell1 の番号と最長一致信号の組が入力される。最長一致信号から、最長一致記号系列が探索されたデータか否かを判断。適当なデータを次段へ出力。
  (a) 各 cell2 は二分木に構成されている。よって cell2 部に入力された信号が出力されるまでに $\lceil \log_2 N \rceil$ ステップ要する。
  (b) 各 cell2 はシストリックに動作する。
  (c) 最終的に出力されたデータから、そのデータの最長一致信号が 1 であるものが正しい出力符号となる。

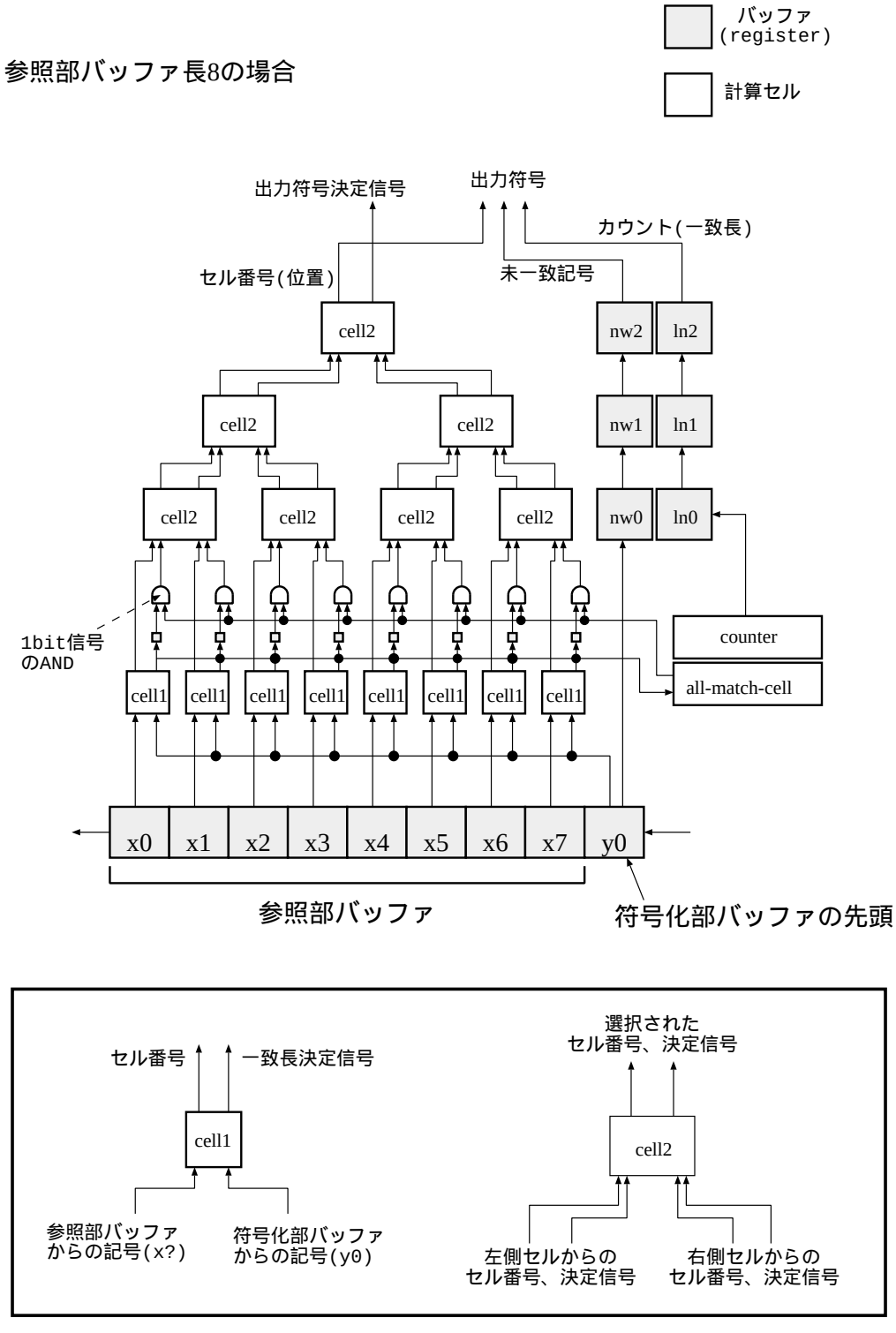


図 3.1: PAHL-C の基本構成 (N = 8)

(ii) (3) での一致判定終了時点でのカウンタ値 (最長一致長値)、符号化部バッファの先頭記号 (未一致記号) を、$\lceil \log_2 N \rceil$ 分の遅延をかけて出力。これにより最終的に出力される一致位置データと一致長、未一致記号の整合性がとられる。

(i)、(ii) は並列に動作する。

符号化の最初に (1) が実行され、その後全てのデータ (記号系列) が符号化されるまで (2)、(3)、(4) が実行される。一つの符号が生成されるために、始めに (2) が実行され、次に一致長探索終了まで (3) が実行される。その間 (4) は継続して実行されている。一つの符号が生成されるのに要する計算回数 (計算量) は (2)、(3) で要する計算回数であるので、$1 + (\mathrm{L} + 1) = \mathrm{L} + 2$ となる。

PAHL-C は次の特徴を持つ高速 LZ77 符号化アーキテクチャである。

- 参照部バッファの全位置の記号比較演算 (一致記号系列探索) を並列に実行。
- 全位置での一致記号系列探索が終了したか否かをチェックすることにより、一回の符号化における比較演算回数を最長一致長程度 $(\mathrm{L} + 1)$ に抑える。
- $\mathrm{L} + 1$ の記号系列 (最長一致記号系列 + 未一致記号) を符号化するのに $\mathrm{L} + 2$ 回の動作 (初期化含む) を必要とする。

  $\Longrightarrow$ 一回の動作で約 1 記号を符号化。

3.1.3　PAHL-C の各計算セルとその機能

[1] 一致判定セル - cell1

一致判定セルでは、入力された参照部バッファ内記号と符号化部バッファの先頭位置の記号との一致判定を行ない、一致長決定信号を出力する。一致長決定信号とは、その時点での比較演算が終了した時点で、まだ (自分のセルで) 一致長探索が継続中であるかどうかを表す信号である。初期状態では 0 にセットされており、0 の間は一致記号系列判定継続中と判断される。一致記号系列探索中に一度でも不一致が生じた時点で 1 にセットされ、次の初期化まで保持される (図 3.2)。

全ての cell1 からの一致長決定信号が 1 となった時点、つまり参照部バッファの全ての位置での一致記号系列探索が終了した時点では、その前時点で一致長決定中の cell1(一致長決定信号が 0 のセル) のうちのいずれかが最長一致記号系列を探索したセルとなる。従って、全 cell1 からの一致長決定信号を見ることにより、全 cell1 で一致判定が終了したかどうかを判断することができ、同時にどの位置で最長一致記号系列が探索されたかを判断することができる。これによって符号化毎の最長一致記号系列探索での記号比較演算を $\mathrm{L} + 1$ 回とすることが可能となる。

cell1 での動作は以下の通りである。

(1) 一致長決定信号の初期化

(一致判定継続中 ： 0、 一致判定終了 ： 1、 初期化時は 0)

(2) 記号一致判定

if 記号一致：

前回の一致長決定信号が 0 → 一致長決定信号： 0
前回の一致長決定信号が 1 → 一致長決定信号： 1

if 記号不一致 → 一致長決定信号： 1

一致記号系列探索の最初に (1) が実行される。その後、全ての cell1 の一致長決定信号が 0 となるまで (2) が繰り返される。

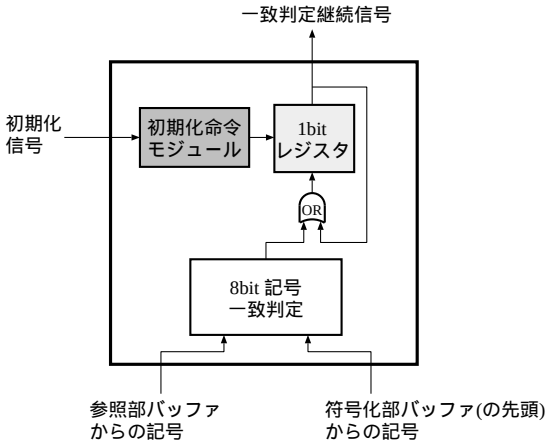


図 3.2: cell1 の内部構成

[2] 最長一致長選択セル - cell2

cell2 では、cell1 部から出力された一致長決定信号 (及びセル番号) と、all-match-cell からの一致判定継続信号とから、どの cell1 で求められた一致記号系列が最長であるかを選択し出力する。

実際に cell2 に入力される信号は、セル番号と最長一致信号と呼ばれる信号となる。最長一致信号は、一致判定継続信号と、その一致判定継続信号の出力される前時点での各 cell1 からの一致長信号の NOT との AND により得られる。最長一致信号が 1 であるデータが、一致記号系列探索終了時に最長一致記号系列の得られた位置の信号となる (図 3.3)。

cell2 での動作は以下の通りである。

(1) 所望データの選択

- 最長一致信号が 1 となる側の信号を次段へ出力

- もし双方が 0 または 1 の場合、左側 (cell1 の番号の小さい側) の信号を次段へ出力

cell2 部は二分木に構成され、シストリックに動作する (木構造シストリックアレイ)。なお、この動作に同期して、一致判定時のカウント値 - 1(一致記号系列長。正確には一致記号系列探索終了時点の前時点でのカウント値)、符号化部バッファの先頭記号がそれぞれの遅延用バッファをシフトしていく。符号出力側に適当な最長一致記号系列判定信号 (値が 1 の信号) が出力された時点で同時に出力された cell1 の番号 (一致位置)、一致記号系列長 (最長一致)、記号 (未一致記号) が、出力符号として適当なデータの組となる。

cell2 部は完全二分木構造であるため、$\lceil \log_2 N \rceil$ の高さを持つ。しかし cell2 部へ入力されるデータは、一致記号系列探索が終了する都度、逐次入力される。つまり、前に入力されたデータが出力されているか否かには関係ないので、cell2 部での計算量は一符号あたり $O(1)$ となる。

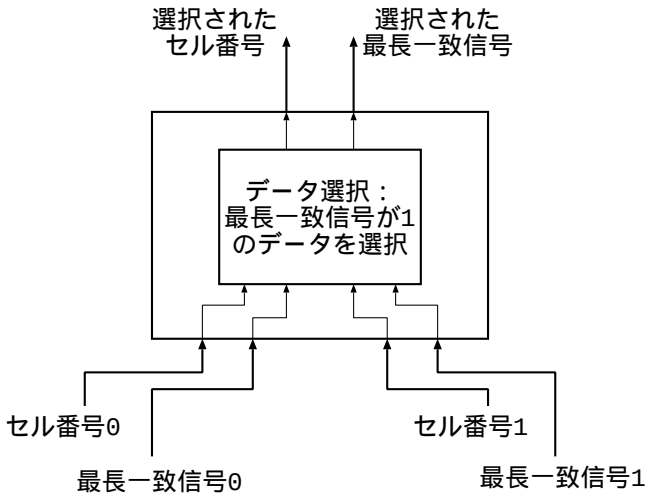


図 3.3: cell2 の内部構成

[3] スライド窓バッファ

スライド窓バッファは、LZ77 符号で用いられる参照部バッファ(辞書バッファ) と、符号化部バッファの先頭記号分のバッファ(8bit シフトレジスタ) により構成されている。このバッファ内の記号系列は、cell1 部における一回の一致判定が実行される度に一記号ずつシフトされる。

[4] 一致記号系列探索終了決定セル - all-match-cell

all-match-cell では、全 cell1 からの一致長決定信号から、全 cell1 での一致判定が終了したかを判断するセルで、N 入力 1 出力の AND 動作を実行する (図 3.4) → 一致判定継続信号を出力。

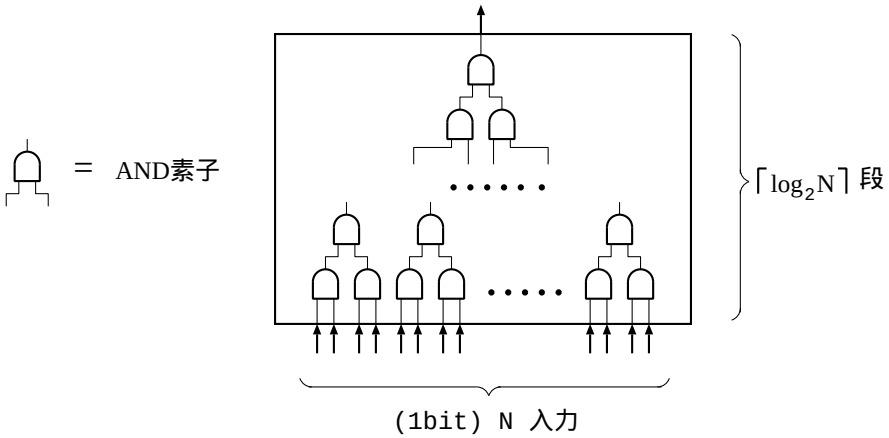


図 3.4: all-match-cell の内部構成 (2 入力 AND による構成例)

[5] カウンタ - counter

カウンタでは、cell1 部において一致判定が継続している間、その一致判定回数をカウントしている。一致判定継続信号が 1 になった時点でそのカウントが終了し、終了の前時点でのカウント値が最長一致記号系列長となる。

カウンタの動作は以下の通りである。

(1) カウンタの初期化

符号化毎に、cell1 部の初期化と同時に実行される (カウント値 : 0)。

(2) カウント実行

cell1 により一致判定が継続している間カウントされ続ける。

(1) は一符号あたりの一致長探索開始時に実行される。一致長決定中は (2) が実行され、カウント値は, カウント毎にバッファ (レジスタ) に保存され、一致判定継続信号が 1 になった時点でバッファに入っているカウント値が、最長一致記号系列長となる。

PAHL-C は以上の計算セル、バッファを組み合せ、それが並列動作することにより、一符号あたり L + 2 回の計算で符号化を実現している。

3.1.4　PAHL-C の計算量

LZ77 符号化において一つの符号を生成するのに必要な計算量は、最長一致記号系列探索における計算量に依存する。PAHL-C では、一致記号系列探索に相当する部分 (cell1 部) と、求められた一致記号系列から最長のものを選択する部分 (cell2 部) とにその処理を分割している。cell2 部での一符号あたりの計算量は O(1) となるので、一符号あたりの計算量は cell1 部に依存することになる。

cell1 部において、一つの符号を生成するのに必要な一致記号系列長を得るのに必要な計算回数 (比較演算) は L + 1 回である。これに初期化に 1 回の動作が必要となる。よって、最長一致長 L を得るのに必要な計算回数は L + 2 回である。

毎ステップに必要な処理時間は、cell1、cell2、counter、all-match-cell、スライド窓バッファに関しては、それぞれの計算セルに必要な処理時間であり、それぞれ

cell1	二つの記号の一致判定 (比較演算) と一致長決定信号の更新
cell2	二つの 1bit 信号による選択
counter	カウント動作
all-match-cell	(1 bit)N 入力 1 出力 AND
スライド窓バッファ	レジスタ間でのデータシフト

に要する時間である。cell2、counter、スライド窓バッファについては、動作自体が単純である、処理時間 (遅延時間) は小さいものと予測される。しかし一致判定部 (cell1 部) では、cell1 の構成 (処理) は他の計算セルにくらべ複雑であり、また all-match-cell の動作が逐次的な動作を成す。

以上のことから、PAHL-C の計算量、動作速度に依存するのは一致判定部 (cell1 部 + all-match-cell) であることが予測される。

3.1.5　PAHL-C の性能評価

- **PARTHENON による PAHL-C の論理設計・動作検証**

PARTHENON (パルテノン: Parallel Architecture Refiner THEorized by Ntt Original coNcept) は、NTT の研究所で独自に開発されたハードウェア設計・支援システムである。また、PARTHENON で用いるハードウェア記述言語 SFL (Structured Function description Language) も、同じく NTT で開発された言語である。

PARTHENON を使用することで、ハードウェア記述言語 SFL を用いたトップダウン設計が可能となる。そして、どのような ASIC (Application SpecificIntegrated Circuit) の製造メーカ、製品系列に対しても、それに対応したセル・ライブラリさえ用意すれば、その製品系列向けに最適化された論理合成を実施することができる (セル・ライブラリを記述する言語が用意されている)。

PARTHENON の出力となるネットリスト (部品間の接続情報) を、各社の配置・配線プログラムへ入力し、その後の工程を踏むことで、極めて短期間で ASIC を開発することが可能となる。また、ネットリストを各社の FPGA (Field Programmable Gate Array) マッピング・ツールへ入力することで、FPGA 化も可能である。つまり、自分だけのオリジナル LSI を、短期間で、安価に設計できる。

本研究での、PARTHENON を用いた性能評価の手順を図 3.5に示す。

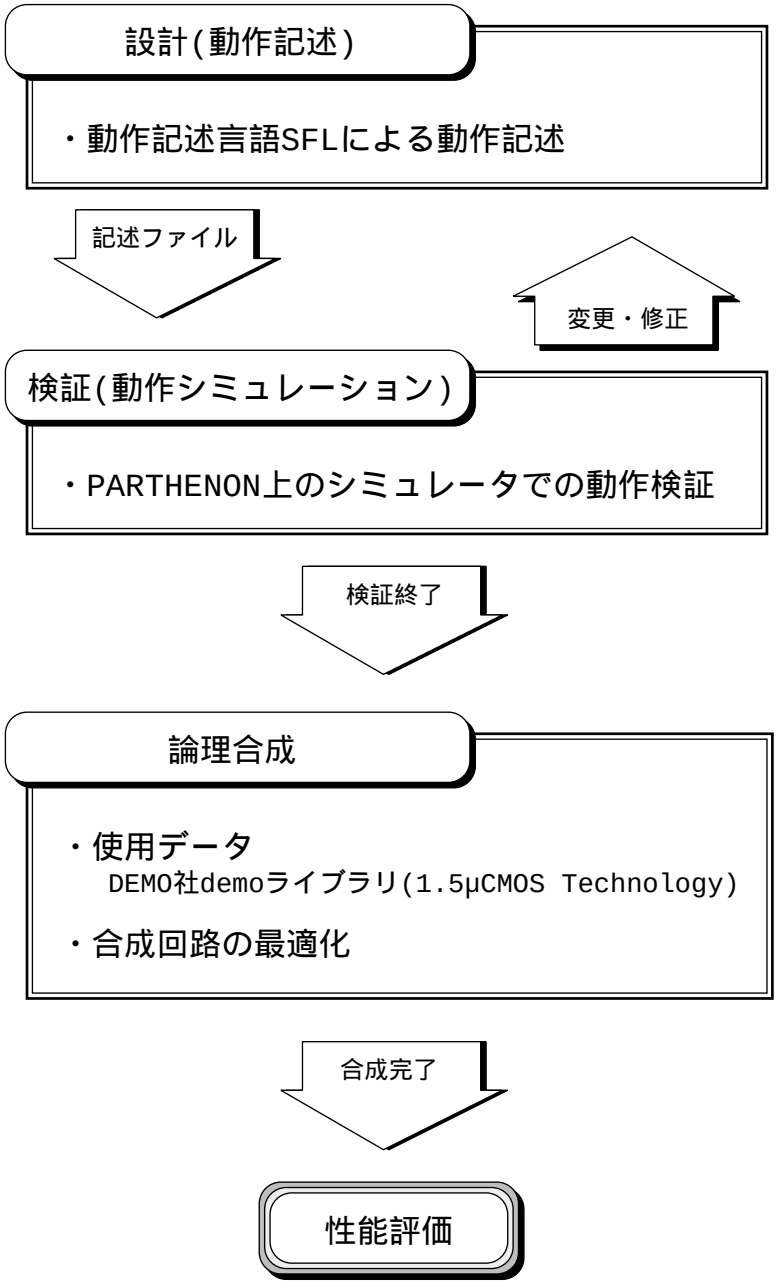


図 3.5: PARTHENON による性能評価の手順

3.1.6　PAHL-C の論理合成結果

PARTHENON を用いての論理設計、動作検証、論理合成により、実現される回路の規模、動作速度等についての評価を行なった。以下に、PAHL-C の消費電力、実装面積、ゲート数、遅延時間、PAHL-C を構成する各計算セルの消費電力、実装面積、ゲート数、遅延時間について示し、PAHL-C の性能について考察する。

各図表での論理合成結果は、全て参照部バッファ(辞書バッファ) 長 N が 8、16、32 での結果である。LZ77 符号において比較的良い圧縮率を実現できるパラメータとしては、N = 4K ないし 8K、符号化部バッファ M = 32 程度であるということが確認されており、現在流通している (LZ77 符号が利用されている) データ圧縮アプリケーションでもその程度のパラメータ値が採用されているようである。しかし本研究では、PARTHENON を運用する環境等の問題により、N、M ともに 8、16、32 の場合での論理合成のみ実現可能であった。しかし、PAHL-C の構成から考えて、ゲート数、消費電力、実装面積は N に比例するものと考えられる。また遅延時間に関しては、各処理部、各計算セルが並列に動作するアーキテクチャであるので、各計算セル単位での最大遅延時間により推定可能である。従って、本研究で得られた論理合成結果から、実用上のパラメータ値 (N = 4K、M = 32 程度) での回路規模のおおよそを求めることができる。

[1] **PAHL-C の消費電力**

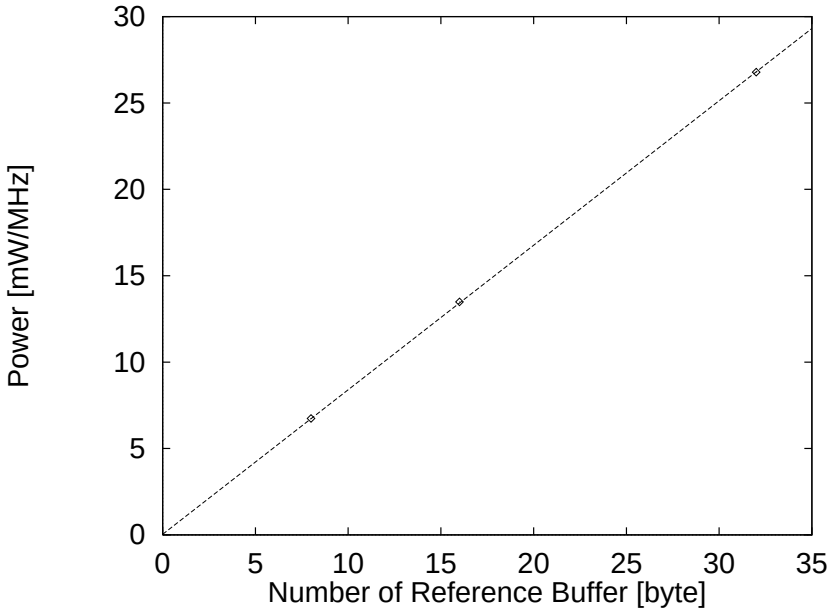


図 3.6: PAHL-C の消費電力

図 3.6より、PAHL-C での消費電力は、予測と同様に参照部バッファ長 N に比例して増加することが示された。実際には個々のセル数等は、

O(N) で増加

- 参照部バッファ長
- cell1 数
- cell2 数
- all-match-cell の (1 ビット信号) 入力数

O($\lceil \log_2 N \rceil$) で増加

- カウント値、未一致記号用遅延バッファの段数

固定

- counter 数
- all-match-cell 数

となり、cell2、counter の入出力ビット幅は $\lceil \log_2 N \rceil$、$\lceil \log_2 M \rceil$ である。また、PARTHENON での論理合成時の (論理式等の) 最適化などもあり、実際には、N の増加に従って、比例関係ではなく N の増加に伴い増加率は少しずつ小さくなっている。しかし、実際の論理合成で得られた結果の範囲ではほぼ比例関係を示している。従って、消費電力 (回路規模) は N に比例するものと考えることができる。以上のことを考慮して、N がそれぞれ 1K、4K、8K の場合の (予測) 消費電力を表 3.1に示す。

表 3.1: PAHL-C の消費電力

参照部バッファ長 N	1K	4K	8K
消費電力 [W/MHz]	0.8	3.2	6.4

[2] PAHL-C の実装面積

PAHL-C の実装面積についても、図 3.7で示されるように消費電力の場合と、定性的には同様の結果を得た。以上にことを考慮して、N がそれぞれ 1K、4K、8K の場合の (予測) 実装面積を表 3.2に示す。

表 3.2: PAHL-C の実装面積

参照部バッファ長 N	1K	4K	8K
実装面積 [mm^2]	200	800	1600

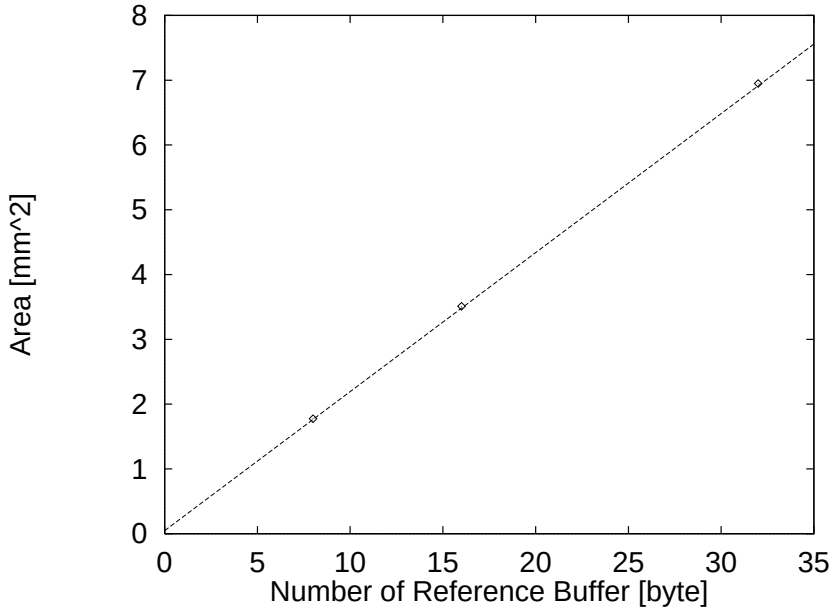


図 3.7: PAHL-C の実装面積

[3] PAHL-C のゲート数

PAHL-C のゲート数についても、図 3.8で示すように消費電力、実装面積の場合と定性的に同様の結果が得られた。次に N がそれぞれ 1K、4K、8K の場合の (予測) ゲート数を表 3.3に示す。

表 3.3: PAHL-C のゲート数

参照部バッファ長 N	1K	4K	8K
ゲート数	280K	1120K	2240K

[4] PAHL-C の遅延時間

PAHL-C では、各計算セル、スライド窓バッファ等が並列に動作するアーキテクチャである。従って PAHL-C としての動作速度は、各計算セル単位の遅延時間に影響されることになる。PAHL-C としての動作速度は、以後に示す各計算セルの遅延時間から予測している。

[4.1] cell1 の遅延時間

cell1 では、入力は参照部バッファ及び符号化部バッファからの記号 (それぞれ 8[bit]) で、出力は cell1 の番号 (参照部バッファ上の位置 : $\lceil \log_2 N \rceil$ [bit]) と一致長決定信号 (1 [bit])

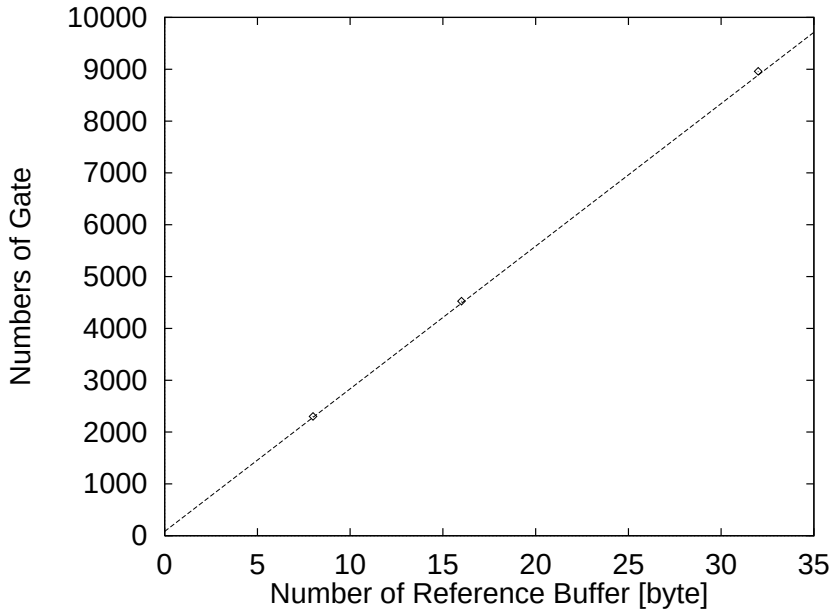


図 3.8: PAHL-C のゲート数

である。cell1 の番号を示す信号のビット幅のみ N に依存するが、この信号は必ずしも各 cell1 から出力させる必要はなく、cell2 部の最下段 (cell1 からの信号を受け取る段) に対して、cell2 の外部から所望の信号を入力すればよいものである。従って cell2 の遅延時間は、スライド窓のパラメータ N、M にかかわらず一定となる。

[4.2] cell2 の遅延時間

cell2 での入力データは、各 cell1 からの前時点での一致長決定信号の NOT と、all-match-cell からの一致判定継続信号との AND (1 [bit]) と cell1 の番号を示す信号の組が二つ ($2\lceil \log_2 N \rceil + 2$ [bit]) で、出力が入力と同様の信号の組 ($2\lceil \log_2 N \rceil + 2$ [bit]) となる。よって入出力ビット幅は N に依存することになる。しかし、N に依存するのは扱うデータのビット幅のみであるため、N が 500 程度以上では N に対して遅延時間はそれほど増加せず、ほぼ一定と考えることができる。

[4.3] all-match-cell の遅延時間

all-match-cell では、入力 N [bit]、出力 1 [bit] の AND 動作を行なう。PAHL-C は、cell2 部を完全二分木に構成することを前提としているため、N は 2 のべき乗の値となる (実際に利用されている N も、ほぼこれに従う)。この条件での論理合成結果から、N が 500 程度以上では遅延時間は 20 [ns] 程度で、大きく増加しないとい結果を得た。ただし、N が 2 のべき乗から外れた値の場合、グラフで示される値よりも大きい遅延時間が求められる場合がある。これは、

- N が 2 のべき乗では、all-match-cell は AND 素子が完全二分木で構成されるが、N が 2 のべき乗から外れた場合、完全二分木では構成できない (パスによって遅延時間が異なる)。
- 論理合成時において、遅延時間が最適になるように、適当な多入力 (入力数 2、3、4) の AND 素子を選択する

という条件により、N が 2 のべき乗から外れた場合、適当な AND 素子による効率的な合成が困難になるためであるものと考えられる。

[4.4] スライド窓バッファ及び cell2 部でのデータ遅延用バッファの遅延時間

これらのバッファに要する遅延時間は、シフトレジスタのデータシフトに要する遅延時間となる。

[4.5] counter の遅延時間

counter では、入出力ともにビット幅 $\lceil \log_2 M \rceil$ [bit] である。これは M に依存しているが、前で述べたように実験的には M = 32 程度で十分であると言われているので、ほぼ一定であるものと考えてよい。

各計算セル別の論理合成により得られた性能結果から、

- N が 500 程度以上では、N の増加に対し遅延時間はそれほど増加しない。

ことが確認された。アーキテクチャの提案当初は、全ての計算セルが並列に動作するものを目指していたが、本研究で提案している PAHL-C では、cell1 と all-match-cell との間での並列動作を実現するに至っていない。このため、PAHL-C の動作速度を考える場合、遅延時間的には、cell1 部 と all-match-cell は一連の動作をする部分として考えなければならない。図 3.9に、cell1、cell2、all-match-cell、counter、cell1 + all-match-cell の最大遅延時間を示す。

[5] 論理合成結果による PAHL-C の予測性能

論理合成により得られた PAHL-C の消費電力、実装面積、ゲート数、また PAHL-C を構成している各計算セルにおける遅延時間から、所望のパラメータ値 M、N での PAHL-C のおおよその性能を予測することができる。次に、符号化部バッファ長 M = 32、参照部バッファ長 N = 1K、4K、8K の場合についての予測性能を表 3.4に示す (1.5 μ CMOS Technology Data による)。

なお PAHL-C では、L + 1 の記号系列を符号化するのに要する計算回数が L + 2 回であるということから、ほぼ 1 ステップに 1 記号が符号化されるものと考えることができる。従って、得られた遅延時間から容易にスルー・レート (1 秒あたりの符号化データ容量 (記号系列長) を算出することができる。

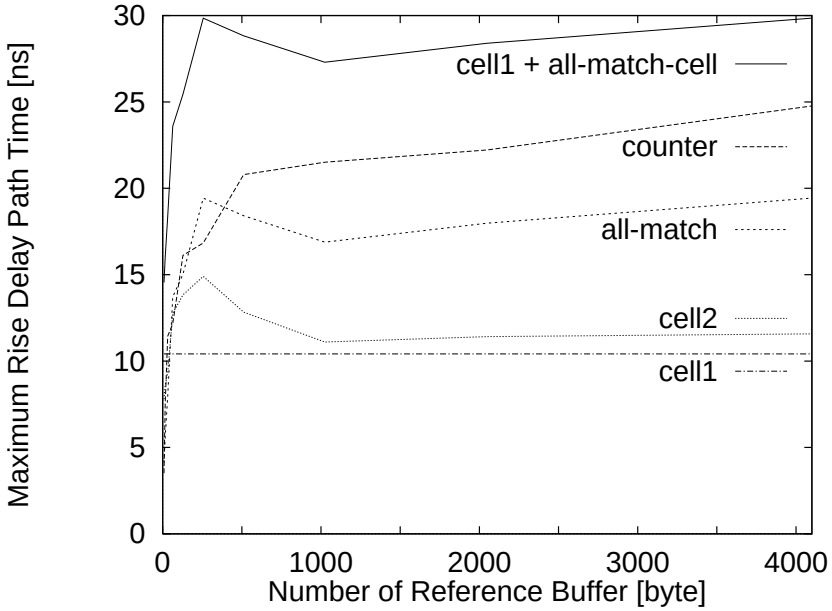


図 3.9: cell1、cell2、counter の最大遅延時間

表 3.4: PAHL-C の予測性能

参照部バッファ長 [byte]	1K	4K	8K
消費電力 [W/MHz]	0.8	3.2	6.4
実装面積 [mm^2]	200	800	1600
ゲート数	280K	1120K	2240K
最大遅延時間 [ns]	50	50	50
スルー・レート [Mbyte/sec]	20	20	20

3.1.7　PAHL-C の実装における課題

PARTHENON による論理設計 (動作記述)、動作検証、論理合成により、LZ77 符号における生成符号の統計的特徴を利用しで計算量削減を実現する、高速 LZ77 並列処理アーキテクチャ - PAHL-C を実現できることを示し、その場合のおおよその性能を求めることができた。得られた予測性能から、PAHL-C では実用上の N、M において重要な点となることとして、

(1) 一致判定継続終了の決定 (all-match-cell) での遅延時間

(2) 実装するにあたっての回路規模 (特にゲート数)

が挙げられる。

(1) に関しては、全ての計算セル (cell1、cell2、counter) 及びスライド窓バッファは並列動作し、より高速処理を実現しているが、all-match-cell は 、cell1 部とはシリアルに動作することになり、しかも全 cell1 からの一致長決定信号を得て、全信号の AND を実行する部分であるので、この部分での遅延時間が PAHL-C での動作速度に大きな影響を与えるものとなるのではないかと考えられる。しかし、本研究での実際の論理合成結果からは、N = 500 程度以上の場合、遅延時間はほぼ一定になるという結果を得ている。また、この部分における多入力 AND (または OR) については Wired-AND (または Wired-OR) により、単なる結線により実現されると、より高速な実行が可能になるものと考えられる。

(2) に関しては、論理合成で得られた予測性能から、その回路規模がかなり大規模になることが明らかになった。この大きな理由としては、回路内にスライドバッファを持つことが挙げられる。実際用いられるゲート数の約半数がスライド窓バッファに要するシフトレジスタに用いられるゲートであることも確認している。

3.2　高速 LZ77 復号化並列処理アーキテクチャ - PAHL-D

3.2.1　LZ77 復号化アルゴリズム内の並列性

LZ77 復号化の主な処理となるのは、符号内の一致位置、一致長情報より、参照部バッファ(辞書バッファ) 上の一致位置から始まる一致長に相当する記号系列を得る処理となる。

LZ77 復号化にでは、単に辞書内の所望長の記号系列を出力することが処理の本質となる。そのためソフトウェアとして実現した場合、符号化に比べ復号化は非常に高速に処理できる。これをハードウェアとして実装する場合を考えると、高速処理の手法として挙げられるのが、並列処理技法を用いることである。しかし、以上のような単純なアルゴリズムであり、LZ77 復号化アルゴリズムから並列性を見出すのは困難である。本論文では、LZ77 復号化をハードウェアとして実現するために、次の動作を実現するアーキテクチャについて考える。

(1) 符号から得られた一致長により、参照部バッファ上の一致位置の記号を決定。

(2) 1 回の動作の度に一致位置の記号を出力し、参照部バッファ内記号系列を 1 シフト。同時に、出力された記号を参照部バッファに追加。この動作が L 回繰り返される。

(3) 一致長分の記号系列の出力後、未一致記号の出力と参照部バッファへの追加

以上の動作により、基本的には

- 参照部バッファ上での記号出力位置決定
- 参照部バッファ内の記号系列シフト

のみで復号化を実現することが可能である。また (2)、(3) での復元記号出力時には、同時に辞書更新も実行される。

次に、上記の動作を具体的に実現した高速 LZ77 復号化並列処理アーキテクチャ- PAHL-D について述べる。

3.2.2　PAHL-D の基本構成

PAHL-D は、基本的に図 3.10示す機能により構成されている。

- ○ 参照部バッファ部
  - 長さ N の 8bit シフトレジスタ
- ○ 参照部バッファの所望位置決定部
  - N 入力マルチプレクサ
- ○ 出力記号数制御部
  - 一致長分の記号出力を制御するためのカウンタ

PAHL-D での基本動作次のようになる。

(1) 符号入力

(i) 入力された符号から、最長一致記号系列の一致位置、一致長、それに続く未一致記号の抽出。

(ii) 出力記号数制御用カウンタの初期化 (一致長分に相当する値による)。

(2) 参照部バッファ内記号の出力

(i) N 入力マルチプレクサによる、参照部バッファの一致位置上の記号の出力。

(ii) 参照部バッファ内記号系列を 1 シフト。同時にマルチプレクサから出力された記号の追加 (辞書更新)。

(iii) 出力記号数制御用カウンタの更新。

(3) 未一致記号の復元記号としての出力

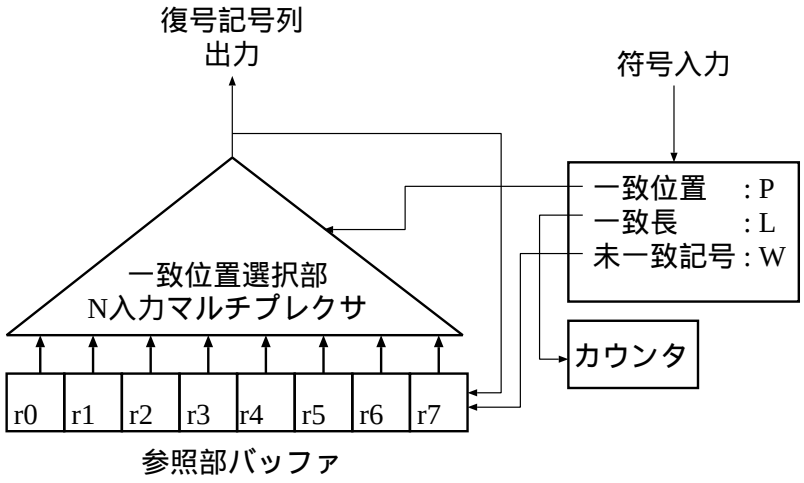


図 3.10: PAHL-D の基本構成 (N = 8)

(i) 符号内の未一致記号の出力。
(ii) 参照部バッファ内記号系列を 1 シフト。同時に未一致記号の追加 (辞書更新)。

以上の (1) ~ (3) の処理により、復号化が容易に実現される。一つの符号により L + 1 の復元記号を得られるが、そのために (1) を実行し、その後 (2) が L 回実行され、(3) を実行し、一符号分の復元記号を得る。また、(2) における (i)、(ii)、(iii)、(3) における (i)、(ii) は共に並列に実行される。従って、 L + 1 の復元記号系列を得るのに L + 2 回の計算回数を要することになり、PAHL-C と同じ計算回数を要する。

3.2.3　PAHL-D の各機能

次に、PAHL-D の各機能について述べる。

[1] 参照部バッファ部
この部分は、PAHL-C の場合と同じく長さ N の 8bit シフトレジスタにより実現される。また参照部バッファは、復元記号出力と同時に実行される。

[2] 参照部バッファの所望位置決定部
参照部バッファの一致位置上の記号を得るため、PAHL-D では N 入力マルチプレクサを用いている。

[3] 出力記号数制御部
長さ L の参照部バッファ内の記号系列を復元記号として出力するため、この記号数を制御するためにカウンタを用いている。具体的には、初期化としてカウンタ値を L に相当す

る値とし、一回の記号出力ごとにカウント値を 1 減少し、0 になった時点で、長さ L の記号系列の出力を終了とし、次の未一致記号出力に移る。

3.2.4　PAHL-D の性能評価

PAHL-D の性能評価についても、PAHL-C の場合と同様に PARTHENON を使用し、SFL による論理設計 (動作記述)、及び PARTHENON 上のシミュレータによる動作検証、及び論理合成を行ない、実現される回路規模、動作速度を求めた。なお、論理合成に用いたデータも PAHL-C の場合と同様、DEMO 社の demo ライブラリ (1.5 μ CMOS テクノロジ) を使用した。

[1] PAHL-D の論理合成結果

PAHL-D の論理合成結果から得られた消費電力、実装面積、ゲート数を図 3.11、3.12、3.13に示す。

PAHL-D の主要部分である参照部バッファ部、参照部バッファの所望位置決定部 (N 入力マルチプレクサ) は、参照部バッファ長 N に比例して増加する部分である。従って、消費電力、実装面積、ゲート数は理論的には N に比例して増加すると考えられていたが、論理合成結果より、消費電力、実装面積、ゲート数はともに N にほぼ比例して増加していることが確認された。従って、任意の N での PAHL-D の消費電力、実装面積、ゲート数を容易に予測することができる。次に実用上の N の値での消費電力、実装面積、ゲート数の予測値を表 3.5に示す。

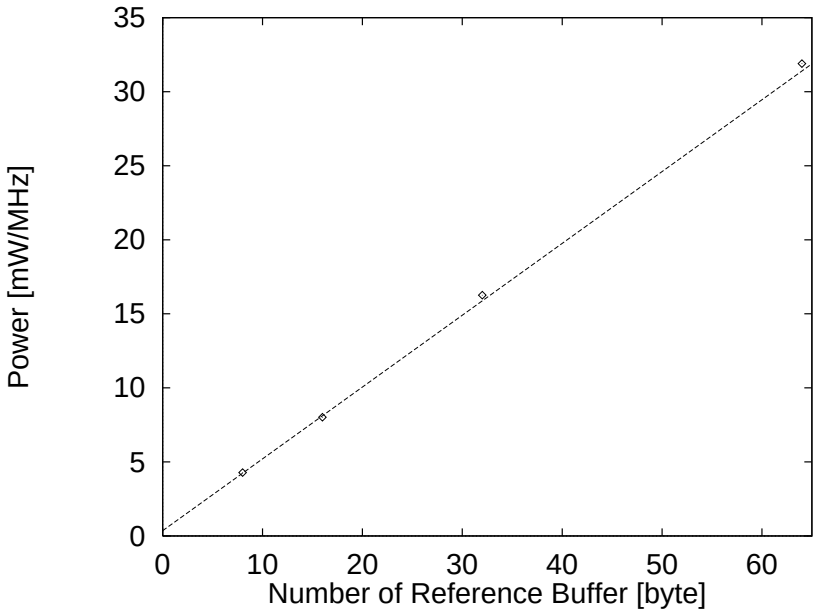


図 3.11: PAHL-D の消費電力

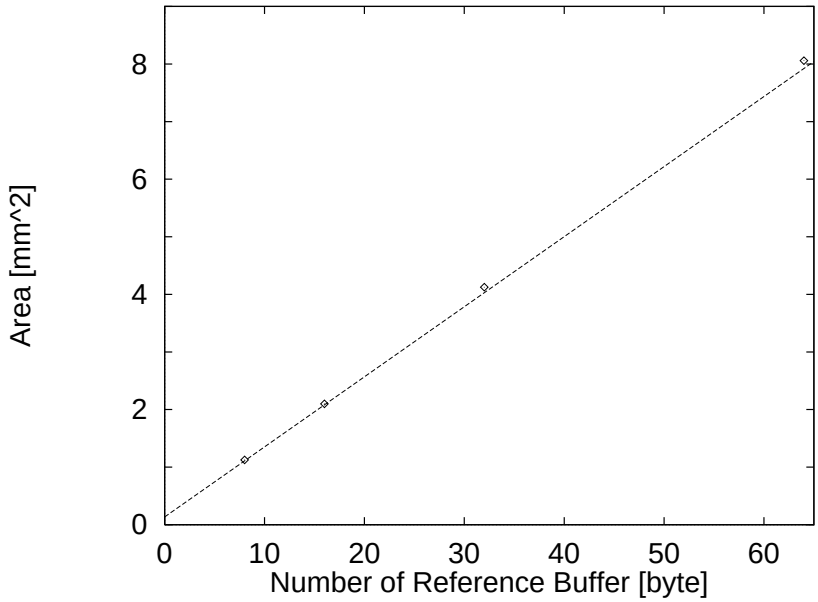


図 3.12: PAHL-D の実装面積

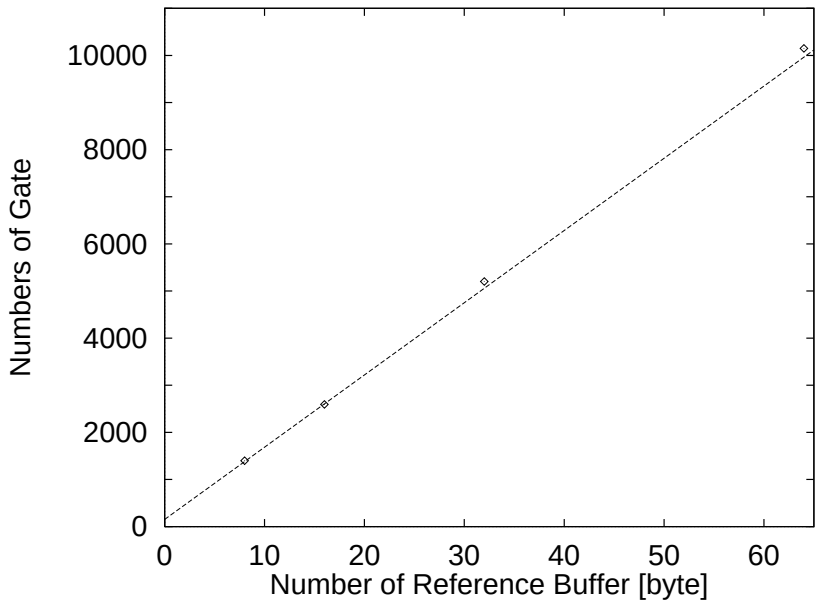


図 3.13: PAHL-D のゲート数

遅延時間については、PAHL-D でも PAHL-C の場合と同様に、各機能部が並列動作する。遅延時間についての考察は、次の PAHL-D の各機能別の論理合成結果で述べる。

表 3.5: PAHL-D の消費電力、実装面積、ゲート数

参照部バッファ長 N	1K	4K	8K
消費電力 [W/MHz]	0.5	2.0	4.0
実装面積 [mm^2]	125	500	1000
ゲート数	160K	640K	1280K

[2] PAHL-D の各機能別の論理合成結果

PAHL-D における主要な構成要素は

- 参照部バッファ部 (8bit シフトレジスタ)
- 参照部バッファの所望位置決定部 (N 入力マルチプレクサ)
- 出力記号数制御部 (カウンタ)

であり、これら三つの構成要素は、参照部バッファ部 ~ 位置決定部と出力記号数制御部の二つの処理が並列に動作する。図 3.14に各機能別の遅延時間を示す。

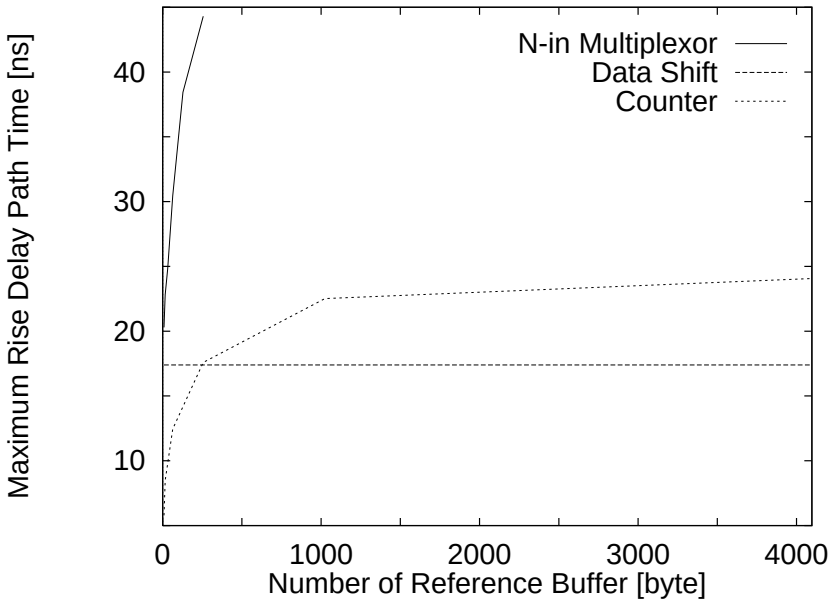


図 3.14: PAHL-D の各機能部の最大遅延時間

各構成要素別の論理合成結果から、参照部バッファ及び出力記号数制御部は、実用上の N (N = 1K 程度 ~) では、参照部バッファは 約 18 [ns] で一定であり、カウンタは約 25 [ns] 前後で大きくは増加せず、ほぼ一定と見ることができる。しかし、位置決定部 (N 入

力マルチプレクサ) の機能については、(こちらも PARTHENON の使用環境等の理由により)N = 1K 程度 ~ の十分な回路規模での論理合成を行なうことが出来ず、より正確な結果を出すことができなかった。位置決定部については、PAHL-C の場合と同様論理合成できた結果からの予測となるが、結果では N とは比例関係ではなく、N の増加につれて遅延時間の増加率は小さくなっている。遅延時間の増加の傾向は、カウンタ部と同じ傾向を示すものと見られ、この点を考慮すると、N = 1K 程度 ~ で約 100 [ns] 以内には収まるものと予測される。

[3] PAHL-D の性能評価

論理合成により得られた PAHL-D の消費電力、実装面積、ゲート数、また PAHL-D を構成している各機能における遅延時間から、所望のパラメータ値 M、N での PAHL-D のおおよその性能を予測することができる。次に、符号化部バッファ長 M = 32、参照部バッファ長 N = 1K、4K、8K の場合についての予測性能を表 3.6に示す (1.5 μ CMOS Technology Data による)。

表 3.6: PAHL-D の予測性能

参照部バッファ長 [byte]	1K	4K	8K
消費電力 [W/MHz]	0.5	2.0	4.0
実装面積 [mm^2]	125	500	1000
ゲート数	160K	640K	1280K
最大遅延時間 [ns]	85	95	100
スルー・レート [Mbyte/sec]	11.8	10.5	10.0

3.2.5　PAHL-D の実装への課題

PAHL-D においても、PAHL-C と同様に PARTHENON による回路設計 (動作記述)、動作検証、論理合成により、任意の M、N における動作性能を予測することができた。しかし PAHL-D の場合においても、実装において次の問題について考慮する必要がある。

(1) 参照部バッファの任意位置選択 (N 入力マルチプレクサ) 部での遅延時間

(2) 実装するにあたっての回路規模 (特にゲート数)

3.3　高速 LZ77 符号化・復号化並列処理アーキテクチャ - PAHL

3.3.1　符号化・復号化アーキテクチャの統合による利点・課題

本研究で提案している PAHL-C、PAHL-D は、それぞれ LZ77 符号化、復号化のみを実行するアーキテクチャである。実際にデータ符号化が行なわれるシステム環境について考えると、

(1) 例えば現在のネートワーク環境では、より高速にデータを相互に交換できることが、ネットワークの効率的運用の重要な点の一つである。

(2) データを相互に交換するようなシステム、または環境で、符号化データ、もしくは符号化処理を扱う場合、符号化または復号化のいずれか一方のみで良いいう事例は少ない (考えられない)。

ということから、符号化されたデータを扱う環境、システムでは符号化 / 復号化双方とも実装されている必要がある。

一般に様々なハードウェアがシステムに実装されるにあたっては、その大きさ (= 回路規模) が大きくなると、それにつれてシステム自体の大きさ、消費電力等がより大きくなってしまう。その結果、より小型・小規模なハードウェアが必要となるが、符号化 / 復号化ハードウェアについて考えると、符号化 / 復号化ハードウェアが効率的な形で統合され、統合されたハードウェアが、符号化 / 復号化ハードウェア双方を共に実装した場合よりも、十分に回路規模が小さくなるものであるなら、符号化 / 復号化ハードウェアの統合は、それを必要としているシステムにとっては相当に有用なものとなる。

次に、PAHL-C / PAHL-D が、PAHL-C の回路規模をほとんど増加させず、かつ PAHL-C / PAHL-D としての性能を維持しつつ統合できることについて述べる。その後、実際に統合された高速 LZ77 符号化・復号化並列処理アーキテクチャ- PAHL を提案し、そのアーキテクチャ、性能等について述べる。

3.3.2　LZ77 符号化・復号化アルゴリズムの共通点

LZ77 符号において、LZ77 復号化過程は LZ77 符号化過程のほぼ反対の処理を実行するものであると考えることができる。つまり、LZ77 符号化過程では、参照部バッファ (辞書バッファ) との最長一致記号系列探索により、最長一致記号系列の先頭位置、一致長、それに続く未一致記号を符号として出力する過程であり、LZ77 復号化過程は、符号内に含まれる一致位置、一致長、未一致記号情報を用い、参照部バッファ内の一致位置から始まる一致長分の記号系列 (これが符号化過程で得られた最長一致記号系列と同じ記号系列である) と未一致記号により復元記号を得る過程である。このことは、LZ77 符号化 / 復号化がとも

に同様の機能を有していることを意味する。具体的には、双方ともスライド窓バッファ(辞書バッファ) を持ち、そのバッファ内の記号系列を扱う。

以上により、LZ77 符号では、符号化・復号化双方で類似した機能を持っており、ハードウェア化した場合においても同様に類似した機能を持つことになり、統合化するにあったて、類似した機能を共有することができるものと考えられる。

3.3.3　PAHL-C、PAHL-D の共通点

PAHL-C、PAHL-D を構成している各 (処理) 機能を図 3.15に示す。

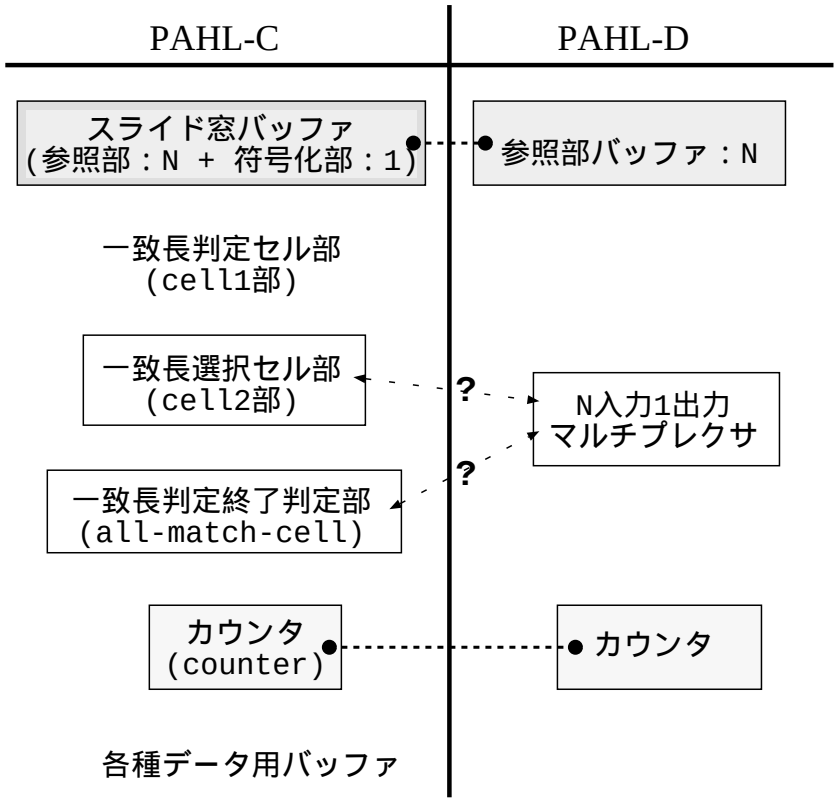


図 3.15: PAHL-C、PAHL-D それぞれの構成要素

図 3.15に示されるように、スライド窓バッファ(= 参照部バッファ)、カウンタは PAHL-C、PAHL-D 双方に同様ものが存在する。また機能部としての構成が同様なものとして、PAHL-D の N 入力マルチプレクサと PAHL-C の cell2 部、または all-match-cell 部がある (いずれも N 入力 - 1 出力)。特にスライド窓バッファ、カウンタはほぼ同じものであり、かつ回路規模 (特にゲート数) に大きく寄与する部分であるので、PAHL-C と PAHL-D を統合するにあたって、この二つの構成要素を共通化することで PAHL-C からの回路規模の増加は大幅に抑えられる。また PAHL-D における N 入力マルチプレクサに関しては、マル

チプレクサのみのゲート数は、各機能別の論理合成結果から、回路全体と比較してかなり小さいものであることを確認している。

以上のことから、PAHL-C に N 入力マルチプレクサ (及び符号化 / 復号化切替にかかわる機能) を付加することで、回路規模をほとんど増加させることなしに、PAHL-C に PAHL-D を埋め込むことが可能である。

3.3.4　高速 LZ77 符号化・復号化並列処理アーキテクチャ - PAHL の基本構成

PAHL-C、PAHL-D を構成するそれぞれの機能のうち、共通な動作 (処理) を行なう部分を共用にすることにより、回路規模をほとんど増加させることなく PAHL-C に PAHL-D の機能を付加することができる。本提案アーキテクチャとしては、PAHL-C をベースとして

- スライド窓バッファ(include 参照部バッファ)
- カウンタ

は共用とし、そこに N 入力マルチプレクサ (位置決定部) と、符号化 / 復号化処理の切替のために必要となる処理 (cell2 部に残っているデータの出力、バッファ、レジスタ類の初期化、等) に要する処理を追加する形式を採用した。従って回路規模としては、N 入力マルチプレクサと符号化 / 復号化の切替に必要な回路、レジスタ分であり、PAHL 全体から比較すると非常に小さいものとなる。

動作性能に関しては、PAHL-C、PAHL-D の動作アルゴリズム、回路構成には直接変更を加えてはいないので、遅延時間等については PAHL-C、PAHL-D の場合と同様の結果になるものと考えられる。

PAHL における符号化・復号化それぞれの動作アルゴリズムは、PAHL-C、PAHL-D の動作アルゴリズムと同じである。また PAHL では、PAHL-C に N 入力マルチプレクサと符号化 / 復号化の切替に必要な処理部を追加した回路構成となるため、PAHL の基本構成は PAHL-C とほぼ同様となる。

動作アルゴリズムは、

(1) 符号化 (PAHL-C と同じ)

(2) 符号化 / 復号化切替

(3) 復号化 (PAHL-D と同じ)

となる。(1)、(3) はそれぞれ PAHL-C、PAHL-D と同様である。(2) における符号化 / 復号化の切替では、PAHL に対して切替を要求する信号を出し、その後 PAHL 側から切替完了の信号が返された後、次の処理を実行するアルゴリズムとなる。切替信号は、符号化記号系列・復号化される符号が全て入力された後に入力されものとする。具体的には、

(a) 符号化 → 復号化切替

参照部バッファ長8の場合

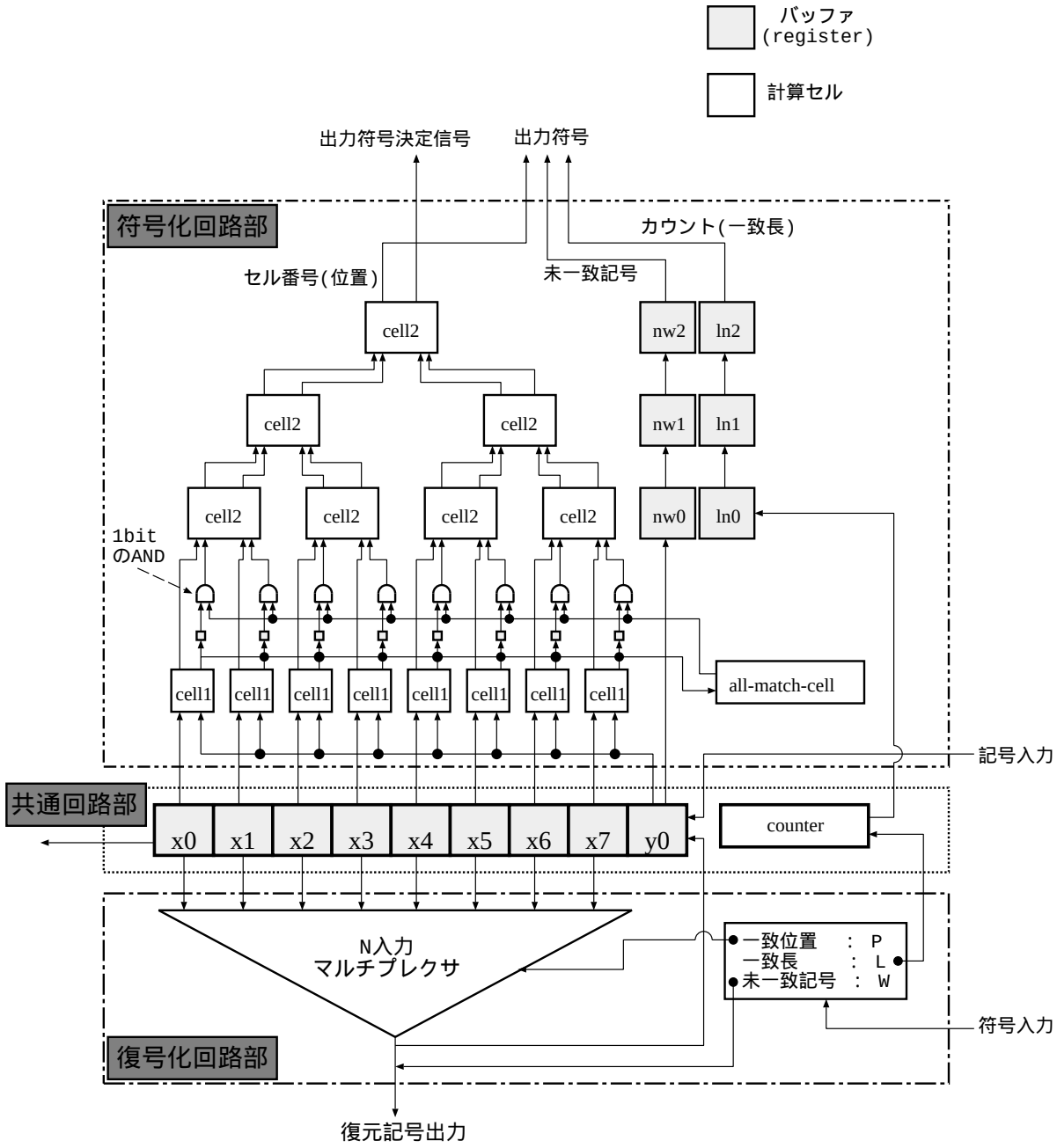


図 3.16: PAHL の基本構成 (N = 8)

(i) 符号化 / 復号化切替信号入力
(ii) cell2 部内符号データ出力
cell2 部及び一致位置、一致長遅延用バッファ部を段数分 ($\lceil \log_2 N \rceil$ 回) 動作させ、cell2 部内のデータを出力する。出力されたデータの中に符号に該当するデータがあった場合、それは出力符号とみなされる。
(iii) 復号化処理のための各初期化
(iv) 復号化開始

(b) 復号化 → 符号化切替
(i) 符号化 / 復号化切替信号入力
(ii) 符号化処理のための各初期化
(iii) 符号化開始

となる。復号化から符号化への切替は容易に実装可能である。符号化から復号化への切替時には、切替信号入力直前までの cell1 部での結果とカウント値、符号化部バッファの先頭位置の記号 (cell2 部に送られるデータ) を出力する必要が生じる。このため、符号化から復号化処理が実行される状態にする間に、cell2 部内にまだ残っているデータを出力させる必要があり、切替信号入力後に cell2 部を $\lceil \log_2 N \rceil$ 回実行させなければならなくなる。この部分が符号化 → 復号化において要する計算となる。

PAHL の動作アルゴリズムが、符号化・復号化においては PAHL-C、PAHL-D と同様であるので、計算量は以下のようになる。

(a) 符号化 ： 一符号あたり $\mathrm{L}+2$ ステップ → $\mathrm{O(L)}$
(b) 復号化 ： 一符号あたり $\mathrm{L}+2$ ステップ → $\mathrm{O(L)}$
(c) 符号化 → 復号化切替 ： $\lceil \log_2 N \rceil$ ステップ → $\mathrm{O}(\lceil \log_2 N \rceil)$
(d) 復号化 → 符号化切替 ： 1 ステップ → $\mathrm{O}(1)$

3.3.5　PAHL の性能評価

PAHL の性能評価についても、PAHL-C、PAHL-D と同様の条件、プロセス、データで性能評価を行なっている。

PAHL を設計するにあたり、符号化 / 復号化両方の設計 (動作記述) が必要になる。それぞれは PAHL-C / PAHL-D と同様に設計されるものになるが、スライド窓バッファとカウンタは共用になる。従って PAHL は、PAHL-C に対し、アーキテクチャの構成要素として新規に位置決定部 (N 入力マルチプレクサ) と符号化 / 復号化切替時処理部の二つが追加される。

3.3.6　PAHL の論理合成結果

論理合成結果から得られた PAHL の消費電力、実装面積、ゲート数を PAHL-C、PAHL-D の場合と共に図 3.17、3.18、3.19に示す。

各図は同様の傾向を示している。その結果から、PAHL の回路規模は (回路全体としては) PAHL-C より多少増加しているものの、ほとんど同様であり、回路規模の増加を抑えて符号化・復号化アーキテクチャを容易に統合する (PAHL-C に PAHL-D の機能を埋め込む) ことが可能であることが確認された。

統合による回路規模の増加は、主に (復号化過程で利用される)N 入力マルチプレクサと、符号化 / 復号化切替時の処理 (処理中のデータの出力、初期化など) に必要な回路分である。N 入力マルチプレクサ部は処理が単純なため回路規模は全体に比べ相当に小さいものである。また符号化 / 復号化切替に伴う処理に関しては、主に符号化→復号化に際して、切替信号入力時にまだ cell2 部内に残っているデータの出力となる。cell2 部に入力された信号が出力されるまでに $\lceil \log_2 N \rceil$ ステップ必要であるので、切替信号入力時から復号化処理状態に遷移させるまでに、cell2 部を $\lceil \log_2 N \rceil$ ステップ動作させる必要がある。ここで提案している PAHL では、$\lceil \log_2 N \rceil$ ステップの動作回数を制御するためのカウンタを別個に設けている。このカウンタが、符号化→復号化切替時の処理で付加される機能である。N 入力マルチプレクサ、カウンタはそれぞれ O(N)、O($\lceil \log_2 N \rceil$) で増加する部分となるが、双方とも PAHL 全体と比べるとその回路規模、及び N に対しての増加は小さいものであることを確認している。従って、N が実用上の大きさ (N = 1K ~) においても PAHL の回路規模は PAHL-C とほぼ同じ程度であるものと予測される。

遅延時間については、符号化・復号化処理それぞれについては PAHL-C、PAHL-D と同じ動作であり、アーキテクチャ自体もほぼ同じである (符号化 / 復号化切替信号の扱いに伴い、入力信号を扱う部分に若干の変更 (追加) がある)。構成している機能、計算セルも PAHL-C、PAHL-D と同じものを用いているので、処理時間 (遅延時間) については、PAHL-C、PAHL-D に関しての議論がそのまま適用できる。

3.3.7　PAHL の予測性能

論理合成の結果から、N = 1K の場合についての PAHL の予測性能を、PAHL-C、PAHL-D と共に表 3.7に示す。予測性能では、PAHL の回路規模は PAHL-C に対し 1.2 倍程度と見積られる。この結果により、PAHL-C と PAHL-D を合せた規模よりも十分に小さいことが確認された。

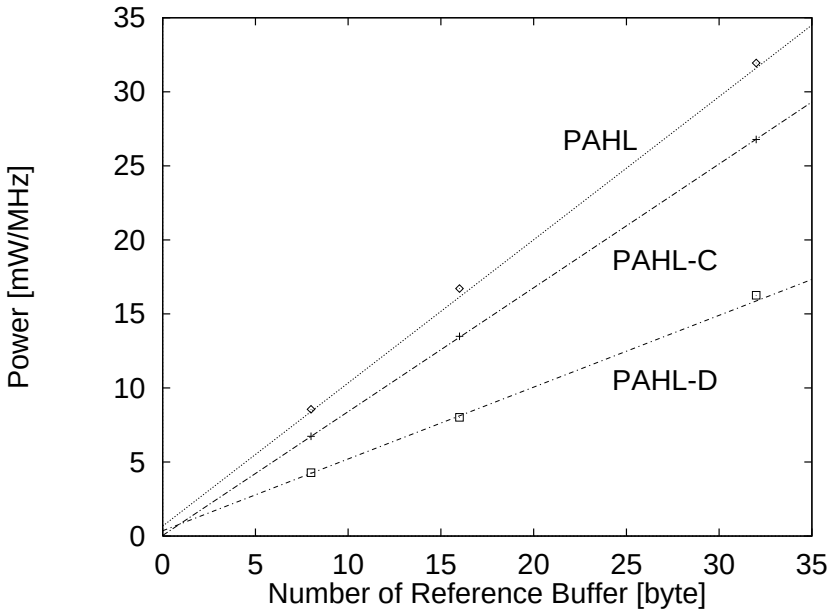


図 3.17: PAHL 及び PAHL-C、PAHL-D の消費電力

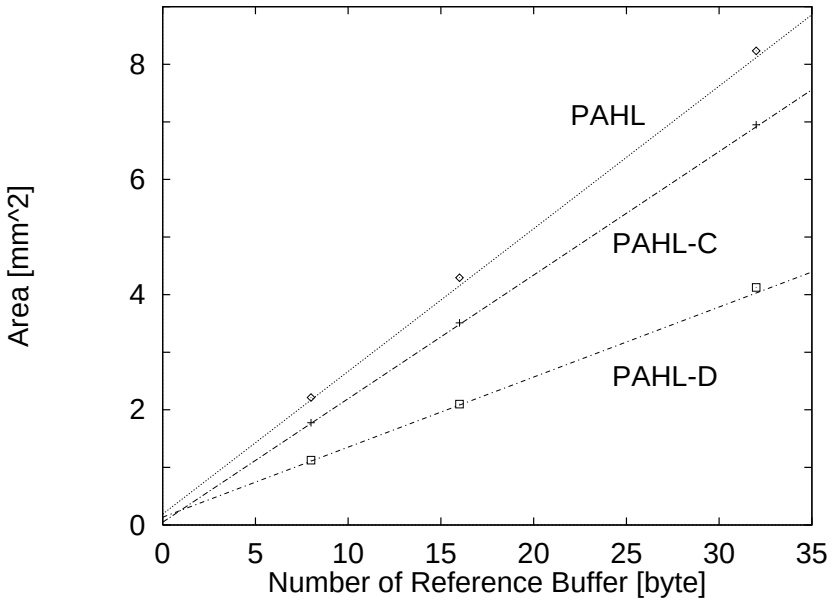


図 3.18: PAHL 及び PAHL-C、PAHL-D の実装面積

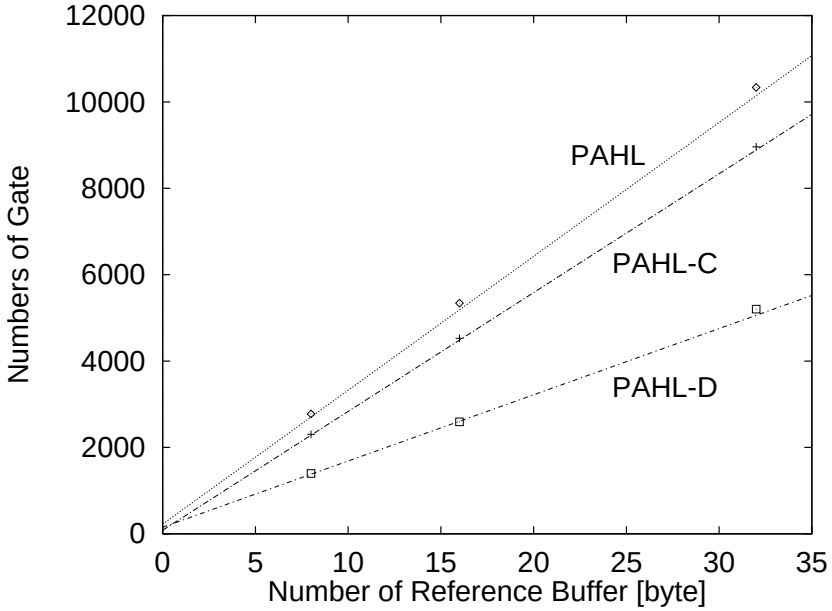


図 3.19: PAHL 及び PAHL-C、PAHL-D のゲート数

表 3.7: 予測される PAHL の性能及び PAHL-C、PAHL-D との比較

	PAHL	PAHL-C	PAHL-D
消費電力 [W/MHz]	0.9	0.8	0.5
実装面積 $[mm^2]$	240	200	125
ゲート数	320K	280K	160K
遅延時間 [ns]	85	50	85
レート [Mbyte/sec]	11.8	20	11.8

第 4 章

高速 LZSS 符号化・復号化並列処理アーキテクチャ - PAHL-LZSS

4.1 LZSS 符号

LZSS 符号は、J. A. Storer と T. G. Szymanski により 1982 年に基本的な考えが提案され、T. C. Bell が 1986 年に具体的な符号化 / 復号化アルゴリズムとして構成したものである。

LZSS 符号は、LZ77 符号の中間符号内の冗長性を取り除くことで、圧縮性能を改善するというアプローチによるものであり、LZ77 符号のバリエーション (改良手法) の中では最も基本的なものである。よって、現在普及、一般で広く利用されている LZ77 符号と呼ばれているもののほとんどは、LZSS 符号が用いられているものと言ってもさしつかえない。

LZ77 符号では、常にポインタ (一致位置 + 一致長) と記号の組み合せが符号語として用いられている。このことは、出力符号としてポインタと記号を交互に出力しているものと考えることができる。しかし、各符号語内に常に圧縮が行なわれていない記号を入れることは明らかに冗長である。そこで LZSS 符号では、1 ビットのフラグを符号語内に設け、ポインタを表す符号語と記号を表す符号語を区別している。

また、LZSS 符号では、未一致記号を常に符号語に入れることをせず、この記号を常に符号化部の先頭に置くことにしている。さらに、符号化部にある記号系列を符号化するにあたり、先頭記号に対応する符号語を用いた場合と、ポインタを表す符号語を用いた場合とで符号長を比較し、短いほうを選択することにしている。

4.1.1 LZSS 符号化

具体的な LZSS 符号の符号化アルゴリズムは LZ77 符号の場合とほぼ同様であり、得られた中間符号から出力符号を生成する部分が変更点となる。

具体的な符号化アルゴリズムを次に示す。

(1) 初期設定

LZ77 符号の場合と同じ。

(2) 符号化

LZ77 符号の場合と同じ。

(3) 符号語の出力

以下の場合分けに従う。

《CASE 1》L ≦ T の場合 (最長一致記号系列長が T 以下の場合)
記号を表す中間符号語〈 f, W 〉を以下のように定義する。
f : 記号を表すフラグ。常に 0
W : 符号化されていない最初の記号

実際の符号語形式は、f (1 ビット)、W (記号を表現するのに適当なビット数) を並べたものとなる (図 4.1(a))。

《CASE 2》L > T の場合 (最長一致記号系列長が閾値 T より大きい場合)
ポインタを表す中間符号語〈 f, P, L 〉を以下のように定義する。
f : ポインタを表すフラグ。常に 1
P : 最長一致記号系列の参照部バッファの先頭からの位置
L : 最長一致記号系列長

ここで、L の取り得る値は、$T + 1$ ~ M となるので、実際の符号語は中間符号語〈 f, P, L 〉から、f (1 ビット)、P ($\lceil \log_2 N \rceil$ ビット)、L ($\lceil \log_2 M \rceil$ ビット) の 2 進数表現を順番に並べたものとなる (図 4.1(b))。

(4) スライド窓の更新

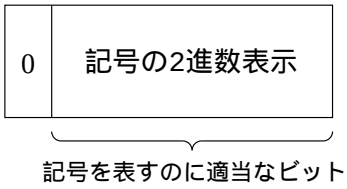
符号化された記号数分だけスライド窓の位置を移動する。実際には

L ≦ T の場合 : $h = 1$
L > T の場合 : $h = L$

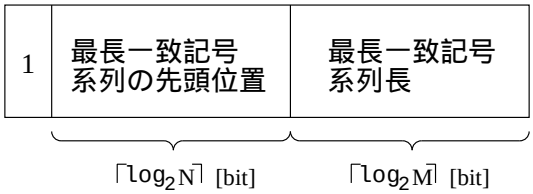
とし、スライド窓の位置を h だけ、記号系列の後に移動する。また符号化位置を示すポインタ i を

$$i = i + h$$

とした後、(2) へ戻る。全ての記号系列を符号化するまで続けられる。



(a)最長一致記号系列長 L が T 以下
の場合の符号形式



(b)最長一致記号系列長 L が T より
大きい場合の符号形式

図 4.1: LZSS 符号の出力符号語形式

4.1.2 LZSS 復号化

具体的な LZSS 符号の復号化アルゴリズムについても、LZ77 符号の場合とほぼ同様である。しかし、入力される 2 種類の符号の形式によって処理が場合分けされる。

具体的な復号化アルゴリズムを次に示す。

(1) 初期設定

LZ77 符号の場合と同じ

(2) 記号列の復号

読み込まれた符号の先頭ビット (フラグ f) により、以下の場合分けに従う。

《CASE 1》f = 0 の場合
符号に続く記号を復元記号として出力し、符号化部バッファの先頭位置にコピーする。

《CASE 2》f = 1 の場合
f に続くビット列から一致位置 P、一致長 L を取り出し、参照部バッファの位置 P から始まる長さ L の記号系列を順番に復号し復元記号系列として出力する。また、符号化部バッファの先頭位置以降にコピーする。

(3) スライド窓の更新

f に応じて

f = 0 の場合 ： h = 1
f = 1 の場合 ： h = L

とし、スライド窓の位置を h だけ、記号系列の後方に移動する。また、符号化位置を示すポインタ i を

i = i + h

とした後、 (2) へ戻る。この処理は、全ての符号を復号するまで続く。

4.2　PAHL から PAHL-LZSS への拡張

LZSS 符号化の基本的処理は、LZ77 符号化と同様の処理である。特に、計算量を最も要する最長一致記号系列探索部は、そのアルゴリズムは LZ77 符号と同じもので実現され、得られた中間符号語から出力符号を求める部分が異なる。つまり LZ77 符号化に、求められる最長一致記号系列長により選択される出力符号形式の条件分岐 (とそれに伴う処理) に相当する処理が追加される。

PAHL-C で行なっている処理は、基本的には LZ77 符号化における中間符号生成である。従って PAHL-C に対し、出力符号形式の条件分岐に相当する回路を追加することにより、PAHL-C を容易に LZSS 符号化に対応可能である。すなわち PAHL-LZSS に拡張することができる。

LZSS 復号化についても、符号化と同様に入力符号の形式により二種類の処理が行なわれることになる。参照部バッファ (辞書バッファ) の一致位置から始まる一致長分の記号系列を復号記号系列として出力する処理 (ポインタ符号が入力された (f = 1) 場合) は同じである。

次に、PAHL を容易に PAHL-LZSS に拡張可能できることを示し、PAHL に必要な機能を追加することにより実現した PAHL-LZSS (PAHL-LZSS-C、PAHL-LZSS-D) について述べる。

4.3　高速 LZSS 符号化並列処理アーキテクチャ - PAHL-LZSS-C

4.3.1　PAHL-LZSS-C の基本構成

PAHL-C で行なわれている処理自体は、LZ77 符号における中間符号生成処理である。従って、基本アーキテクチャは、PAHL-C の基本構成を利用できる。しかし、LZSS 符号

に拡張するにあたり、PAHL-C をそのまま利用しようとした場合、次の問題を解決しなければならない。

◇　LZ77 符号化では、スライド窓の更新は、符号化された記号系列分のスライド窓の移動となる。PAHL-C では、一致長探索とスライド窓のシフトを同時に実行しており、一符号の生成に L + 2 回の計算回数を要する (このときのスライド窓バッファのシフト長は L + 1 となる)。

LZSS 符号化では、得られた L と閾値 T との大小関係によって、二種類の符号が生成される。特に L ≦ T の場合、L がどのような大きさであっても復号化される記号数は 1 である。このとき PAHL-C をそのまま利用しよう とした場合、長さ 1 の記号の符号化にもかかわらず、L + 1 のスライド窓バッファの移動が実行される。従って、次の符号化の実行にあたり、移動しすぎた分 (L) だけスライド窓バッファを元に戻す処理が必要となる。言い替えると、一度スライド窓バッファ(参照部バッファ) から出力された記号系列を L(最大で T) だけ元に戻さなくてはならなくなる。

◇ 二種類の符号を出力するために、符号の出力法 (出力ポート、その他) をどのようにするか考慮する必要がある。

以上の問題を解決し、LZSS 符号化を実現するアーキテクチャ ~ 高速 LZSS 符号化並列処理アーキテクチャ - PAHL-LZSS-C を提案する。次に PAHL-LZSS-C の基本構成を図 4.2に示す。

図 4.2に示すように、PAHL-LZSS-C の基本構成は PAHL-C とほぼ同様である。使用している計算セル (cell1、cell2、all-matchl-cell、counter)、スライド窓バッファは、PAHL で提案、実現したものを使用している。その他に、スライド窓バッファを戻す、つまりスライド窓 (参照部) バッファから一度出ていった記号を元も戻すために必要な一時保存用バッファ(以下、ストックバッファ(バッファ長 T)) と、得られた最長一致記号系列長から符号形式を決定する部分 (符号選択部) が追加される。追加された二つの機能部は、PAHL-LZSS-C の動作制御 (処理状態遷移の制御) に大きく寄与する部分となる。

(1) 初期化-**1**

参照部バッファ(辞書バッファ)、符号化部バッファの先頭位置)、及び一致長選択部 (cell2)、一致位置、一致長伝送、出力用バッファのための各レジスタの初期化。PAHL-C の場合と同じ。

(2) 初期化-**2**

一致長探索部 (cell1)、カウンタの初期化。PAHL-C の場合と同じ。

(3) 一致記号系列探索

全 cell1 部での一致長探索。PAHL-C の場合と同じ。

(i) 各 cell1 による一致判定の並列実行

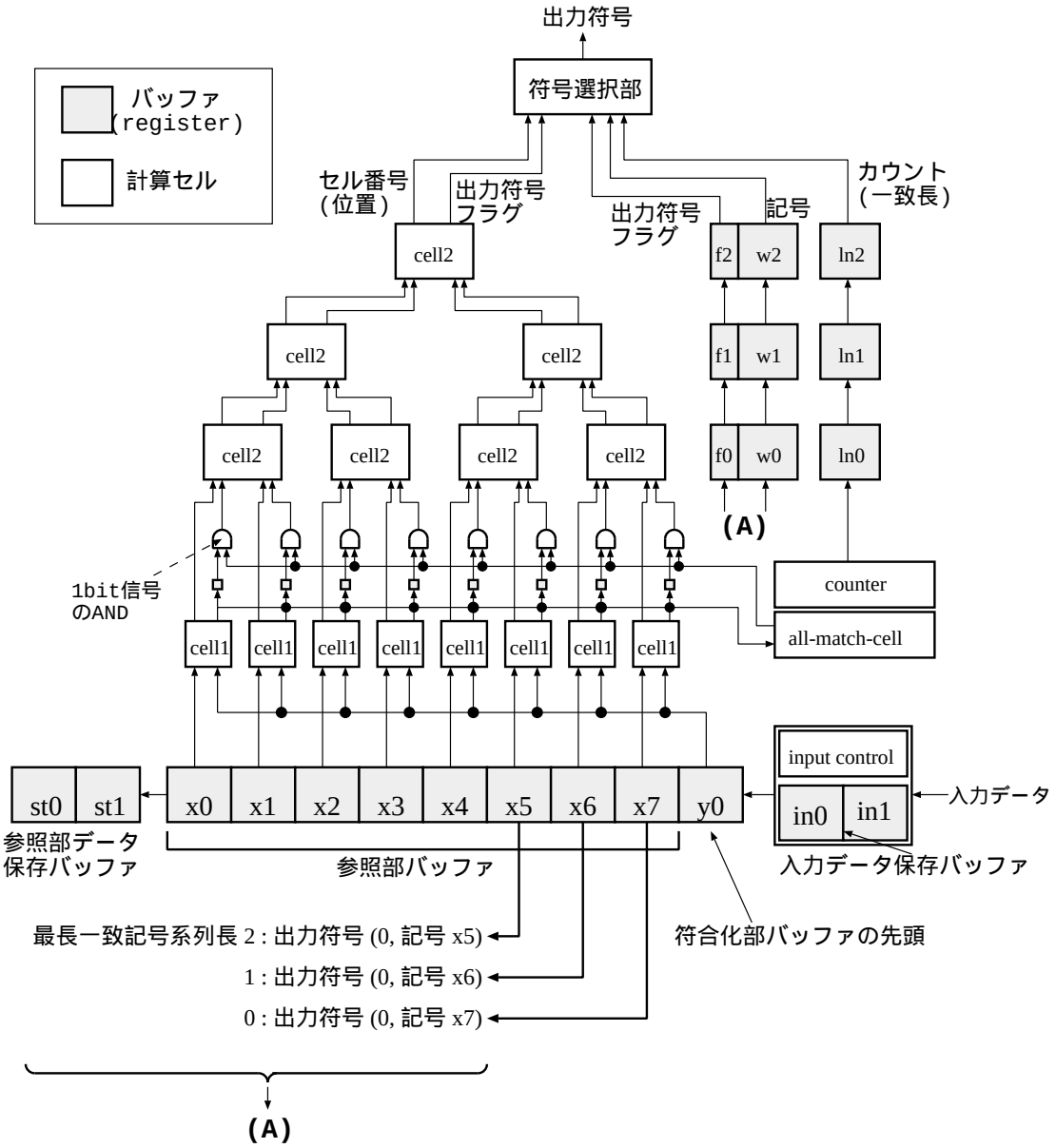


図 4.2: PAHL-LZSS-C の基本構成

(ii) 各 cell1 からの一致長決定信号の (並列) 出力
(iii) 各 cell1 からの一致長決定信号による一致長探索継続の判断 (all-match-cell)
(iv) カウンタの更新
(v) 参照部バッファ、符号化部バッファ内記号の更新
　各バッファ内記号を 1 シフト

(i) ~ (iii) までと (iv)、(v) は並列に実行される。また (i) ~ (v) は一致長探索継続終了の判断が出るまで継続される。

(4) 最長一致記号系列選択

得られた各 cell1 からのデータから所望のデータを選択。PAHL-C の場合と同じ。

(最長一致記号系列位置選択部 ： cell2 部)

○ (3) での処理により得られた (一致判定終了時点の前時点での) 一致長決定信号により、最長一致記号系列の得られた cell1 の番号を選択。この番号が一致位置となる。

(i) 各 cell2 には、前段から二つの cell1 の番号と最長一致信号の組が入力される。最長一致信号から、最長一致記号系列が探索されたものか否かを判断。適当なデータを次段へ出力。

《1》各 cell2 は二分木に構成されている。よって cell2 部に入力された信号が出力されるまでに $\lceil \log_2 N \rceil$ ステップ要する。

《2》各 cell2 はシストリックに動作する。

《3》最終的に出力されたデータから、そのデータの一致長決定信号からそれが正しい符号であるかを判断。

(ii) (3) での一致判定終了時点でのカウンタ値 (最長一致長値)、符号化部バッファの先頭記号 (未一致記号) を、$\lceil \log_2 M \rceil$ 分の遅延をかけて出力。これにより最終的に出力される一致位置データと一致長、未一致記号の同期をとる。

※ (i)、(ii) は並列に動作する。

(5) 出力符号形式の選択及びスライド窓バッファ(内記号系列) の移動

一つの符号生成過程終了ごとに、その時点で得られた最長一致記号系列長により、次の処理が実行される。

PAHL-C で使用されている参照部バッファは、一方向のみのデータシフトが実装されてるが、PAHL-LZSS-C で使用される参照部バッファでは、双方向のデータシフトが実装されている必要がある。

《CASE 1》L ≦ T の場合
　出力符号は記号符号〈0, W〉となり、参照部バッファの適当な位置から直接出力され、符号出力フラグ (フラグ値 1) と共に、cell2 部にある記号符号用遅延バッファ (w_0 ~ 、f_0 ~) に転送される。

この時点でスライド窓バッファは L + 1 だけ移動しているので、スライド窓バッファを L だけ元に戻す (スライド窓バッファ内記号系列を L だけ戻す)。その語、次の符号化処理を実行する。

※ 従って、L = 0 (一致長が 0) ではスライド窓バッファを戻す操作は必要ない。

《CASE 2》L > T の場合

出力符号はポインタ符号〈1, P, L〉となり、cell2 部から出力されることになる (所望のデータが cell2 部から任意ステップ後に出力される)。

この時点でスライド窓バッファは L + 1 だけ移動しているので、スライド窓バッファを 1 だけ元に戻す (スライド窓バッファ内記号系列を 1 だけ戻す)。その語、次の符号化処理を実行する。

符号化における中間符号生成過程 ((1) ~ (4)) は PAHL-C の場合と同じである。その後、一つの符号化での一致長探索終了後に (5) が実行される。

PAHL-LZSS-C では、ポインタ符号は cell2 部から、記号符号は参照部バッファ部から直接出力 (ただし出力ポートは同一) する構成にしている。cell2 部は木構造シストリックアレイを成しているので、データはパイプライン的に移動する。そのためポインタ符号は、(そのポインタ符号が得られた一致長探索過程での) 一致長探索終了から $\lceil \log_2 N \rceil$ ステップ後に cell2 部から出力される。一方、記号符号は一致長探索終了後すぐに参照部バッファから出力される。そこで、PAHL-LZSS-C では出力符号の順番を保つため、記号符号についてもポインタ符号と同様のパイプライン的な遅延を与えている。この遅延により、一符号あたりの計算量を増加させずに符号の出力順を維持している。

以上の構成により、一つの符号を生成するのに必要なステップ数は最大で (符号形式の選択のための閾値：T)

- 記号符号： $2L + 2$ [step]　(L ≦ T)

  (初期化：1 & 一致長探索：L+1 & スライド窓バッファ移動：L)

- ポインタ符号： $L + 3$ [step]　(L > T)

  (初期化：1 & 一致長探索：L+1 & スライド窓バッファ移動：1)

と定義され、一符号あたりの計算量は O(L) となる。

4.3.2　PAHL-LZSS-C の性能評価

PAHL-LZSS-C についても、第 3 章での PAHL 等と同様に、PARTHENON を使用し、動作記述、動作検証、及び論理合成を行ない、実現される回路規模、動作速度を求めた。論理合成に用いたデータも DEMO 社の demo ライブラリ (1.5 μ CMOS テクノロジ) を使用した。次に、PAHL-LZSS-C の消費電力、実装面積、ゲート数、遅延時間ついて、PAHL-C での結果と共に示し、PAHL-LZSS-C の性能について考察する。

[1] PAHL-LZSS-C の論理合成結果 (消費電力)

論理合成により得られた結果 (図 4.3) より、PAHL-LZSS-C についても、PAHL-C に比べて若干増加しているものの、同様の結果、傾向であることが確認された。PAHL-LZSS-C の基本的アーキテクチャ、それを構成している計算セルは PAHL-C と同じものであり、LZSS 符号に拡張するのに若干の機能追加にとどまっているため、消費電力は同様の結果になるものと考えられる。

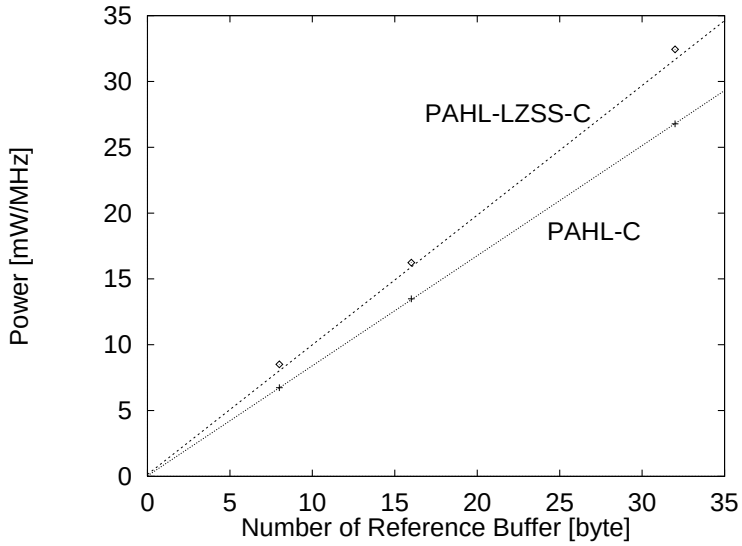


図 4.3: PAHL-LZSS-C の消費電力

[2] PAHL-LZSS-C の論理合成結果 (実装面積)

実装面積についても、消費電力の場合の理由により、PAHL-C の結果とほぼ同様の結果であると見ることができる (図 4.4)。

[3] PAHL-LZSS-C の論理合成結果 (ゲート数)

ゲート数についても、消費電力、実装面積の場合の理由により、PAHL-C の結果とほぼ同様の結果であると見ることができる (図 4.5)。

[4] PAHL-LZSS-C の論理合成結果 (遅延時間)

PAHL-C については、動作速度に関しては各計算セルの遅延時間に依存するものとして議論した。PAHL-LZSS-C においての追加機能は

- 符号選択部

である (他に記号一時ストック用バッファがあるが、これはスライド窓バッファを入力・出力部にそれぞれ T だけ追加したものとなる)。

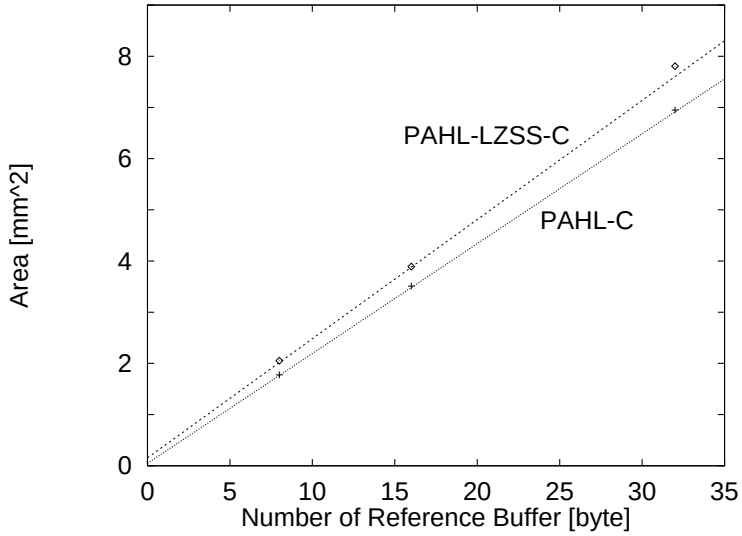


図 4.4: PAHL-LZSS-C の実装面積

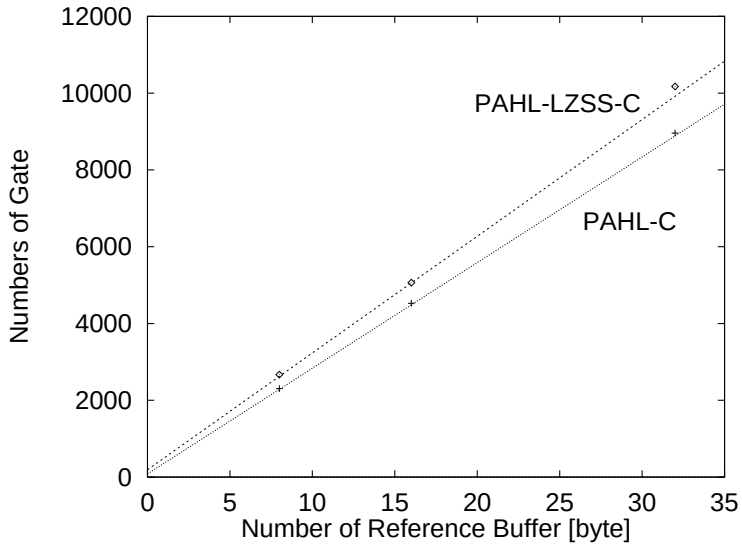


図 4.5: PAHL-LZSS-C のゲート数

符号選択部では、入力される符号候補データの持つ出力符号フラグ、符号形式フラグ (符号形式を表すフラグ) から、所望の符号を選択するものである。従って、二つのフラグ値 (各々1 ビット信号) により判断する部分であるので、他の計算セルに比べて、小規模で遅延時間も小さい部分である。以上により、PAHL-LZSS-C の動作速度 (遅延時間) についても PAHL-C での場合と同じ議論となる。つまり、PAHL-C でも用いられている各種計算セルの動作速度について議論することになる。PAHL-LZSS-C における遅延時間については、PAHL-C と同じものと判断できる。

[5] 論理合成結果による PAHL-LZSS-C の予測性能

論理合成により得られた PAHL-LZSS-C の消費電力、実装面積、ゲート数、また PAHL-LZSS-C を構成している各計算セルにおける遅延時間から、所望のパラメータ値 M、N での PAHL-LZSS-C のおおよその性能を予測することができる。次に、符号化部バッファ長 M = 32、参照部バッファ長 N = 1K、4K、8K、T = 3 の場合についての予測性能を表 4.1 に示す (1.5 μ CMOS Technology Data による)。

なお PAHL-LZSS-C においても、計算量が O(L) であるということから、一つの記号を符号化するのに必要な計算回数は、最悪でも 2 程度である。従って、得られた遅延時間から容易にスルー・レート (1 秒あたりの符号化データ容量 (記号系列長) を算出することができる。

表 4.1: PAHL-LZSS-C の予測性能

参照部バッファ長 [byte]	1K	4K	8K
消費電力 [W/MHz]	1	4	8
実装面積 [mm^2]	240	960	1920
ゲート数	300K	1200K	2400K
最大遅延時間 [ns]	50	50	50
スルー・レート [Mbyte/sec]	20	20	20

4.4　高速 LZSS 復号化並列処理アーキテクチャ - PAHL-LZSS-D

4.4.1　PAHL-LZSS-D の基本構成

LZSS 符号では、ポインタ符号と記号符号の二種類の符号が存在する。このため、入力される符号形式によって、復号化アルゴリズムについても次の二種類が存在することになる。

- ポインタ符号：

符号内の一致位置 P、一致長 L より、参照部バッファ内の位置 P から始まる長さ L の記号系列を、復元記号系列として得る。

○ 記号符号：

符号内の記号を、復元記号として得る。

この二つの処理は、得られる復元記号が、ポインタ符号の場合に得られる参照部バッファ内に存在している所望の記号系列か、または符号記号の場合に得られる符号内に存在している記号かの違いのみで、記号系列を出力し、スライド窓バッファの更新時に復元記号系列がバッファの最後部に追加されるという処理は同じである。従って、復号化アーキテクチャ自体は LZ77 符号の場合とほぼ同様となる。

追加される機能としては、入力符号内のフラグ値による復号処理の選択部である。記号符号の場合は、符号内の記号を復元記号として出力し辞書更新するだけであり、ポインタ符号な場合は PAHL-D と同じ (未一致記号出力を含めず) である。従って。PAHL-LZSS-D のアーキテクチャは PAHL-D と基本的には同じもので実現することができる。よって構成する機能も同一の機能を利用することができる。

PAHL-LZSS-D は、PAHL-D と同様に、基本的には次に示す機能により構成されている (図 4.6)。

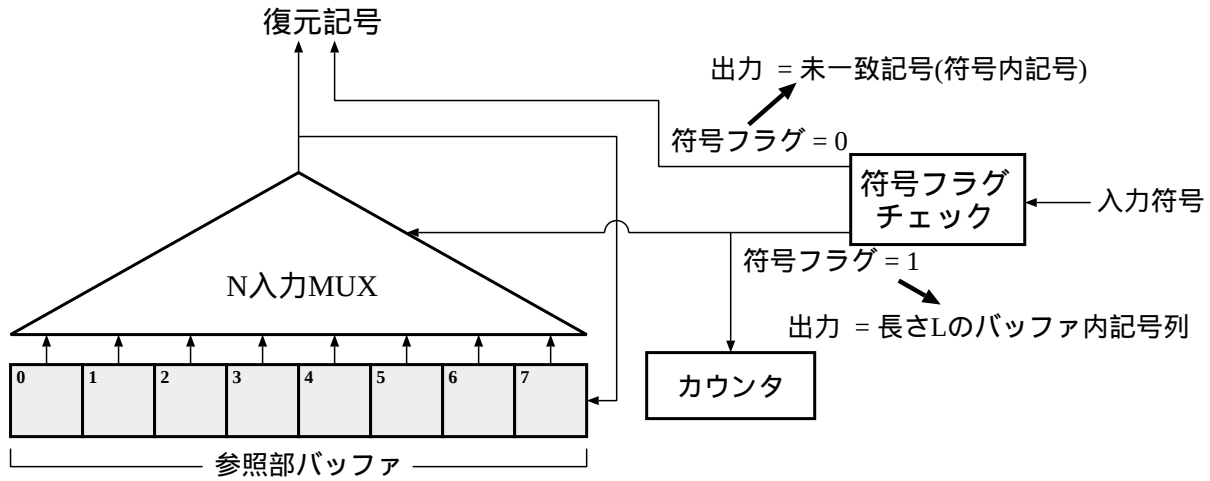


図 4.6: PAHL-LZSS-D の基本構成 (N = 8)

○ 参照部バッファ部

- 長さ N の 8bit シフトレジスタ

○ 参照部バッファの所望位置決定部

- N 入力マルチプレクサ

○ 出力記号数制御部

- 一致長分の記号出力を制御するためのカウンタ

PAHL-LZSS-D の基本動作は以下のようになる。

(1) 符号入力

(i) 符号内のフラグ (先頭ビット) から、符号形式を選択
(ii) フラグ値により、以下の処理を実行する。

《CASE 1》フラグ値が 0 (記号符号の場合)
符号内の記号を抽出。

《CASE 2》フラグ値が 1 (ポインタ符号の場合)
符号内の一致位置、一致長を抽出。

(iii) 出力記号数制御用カウンタの初期化 (一致長分に相当する値による)。

(2) 復元記号系列の出力

◎ フラグ値により、以下の処理を実行する。

《CASE 1》フラグ値が 0 (記号符号の場合)
(i) 符号内記号の出力。
(ii) 参照部バッファ内記号系列を 1 シフト。同時に符号内記号の追加 (辞書更新)。

※ (i)、(ii) は並列動作。

《CASE 2》フラグ値が 1 (ポインタ符号の場合)
(i) N 入力マルチプレクサによる、参照部バッファの一致位置上の記号の出力
→ 復元記号出力
(ii) 参照部バッファ内記号系列を 1 シフト。同時にマルチプレクサにより出力された記号の追加 (辞書更新)。
(iii) 出力記号数制御用カウンタの更新。

※ (i) ~ (ii) と (iii) は L 回並列動作する。

以上の (1)、(2) の処理により、復号化が容易に実現される。一つの符号の復号化に必要なステップ数は、

○ 符号記号 ： 1

(符号入力 & 復元記号出力 & 辞書更新) : 1)

○ ポインタ符号 ： L + 1

(符号入力 : 1 + (((マルチプレクサ動作 & 復元記号出力) + 辞書更新) & カウンタ動作) : L)

となる。復号化に必要な計算量は O(L) となる。

4.4.2　PAHL-LZSS-D の性能評価

PAHL-LZSS-D に性能評価についても、PAHL-LZSS-C と同様の条件、プロセス、データにより、PARTHENON を使用して性能評価を行なっている。

[1] PAHL-LZSS-D の論理合成

PAHL-LZSS-D の論理合成結果から得られた消費電力、実装面積、ゲート数、遅延時間を図 4.7、4.8、4.9、4.10に示す。

PAHL-LZSS-D の主要部分も、PAHL-D と同じく参照部バッファ部、参照部バッファの所望位置決定部 (N 入力マルチプレクサ)、カウンタとなる。消費電力、実装面積、ゲート数の論理合成結果は、PAHL-D の場合とほぼ同じである。この理由も、PAHL-D の場合と同様の理由によるものであると考えられる。

遅延時間については、PAHL-D と PAHL-LZSS-D は基本的に同じ機能により構成されている。従って各機能部あたりの遅延時間は同一であるが、回路全体として実際に動作させた場合、PAHL-LZSS-D のほうが PAHL-D に比べ良いものとなるようである。PAHL-D の場合は一つの符号を復号するために、一致長分の記号系列復元 (参照部バッファ内からの記号系列出力とスライド窓更新) と、未一致記号を復元記号とする処理 (記号出力とスライド窓更新) の二つの処理が行なわれる必要があり、毎符号ごとに処理の切替に相当する操作が必要となる。一方 PAHL-LZSS-D の場合、符号入力時点で、一致長分の記号系列の復元操作か、または符号内の記号の復元記号としての出力操作かが、入力された符号形式により決定され、そのいずれかを実行することになる。PAHL-LZSS-C では、復号化回路における処理が単一 (一つの符号に対しての符号化処理が一種のみ) で実現できるため、そのことが回路全体としての動作に影響を与えるものと考えられる。但し PAHL-LZSS-D もアーキテクチャの基本構成は PAHL-D と同一であり、各処理機能が並列動作することから、性能評価については、各処理機能の遅延時間から求めている。

[2] PAHL-LZSS-D の性能評価

論理合成により得られた PAHL-LZSS-D の消費電力、実装面積、ゲート数、また PAHL-LZSS-D を構成している各機能における遅延時間から、所望のパラメータ値 M、N での PAHL-D のおおよその性能を予測することができる。次に、符号化部バッファ長 M = 32、参照部バッファ長 N = 1K、4K、8K の場合についての予測性能を表 4.2に示す (1.5 μ CMOS Technology Data による)。

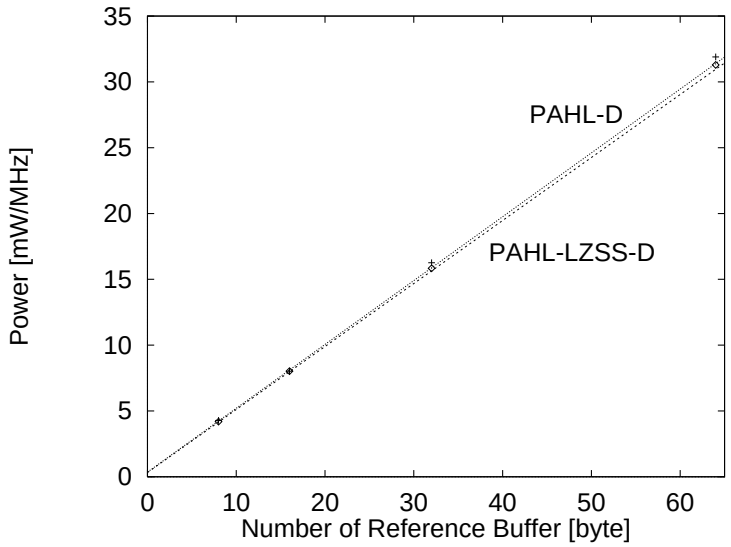


図 4.7: PAHL-LZSS-D の消費電力

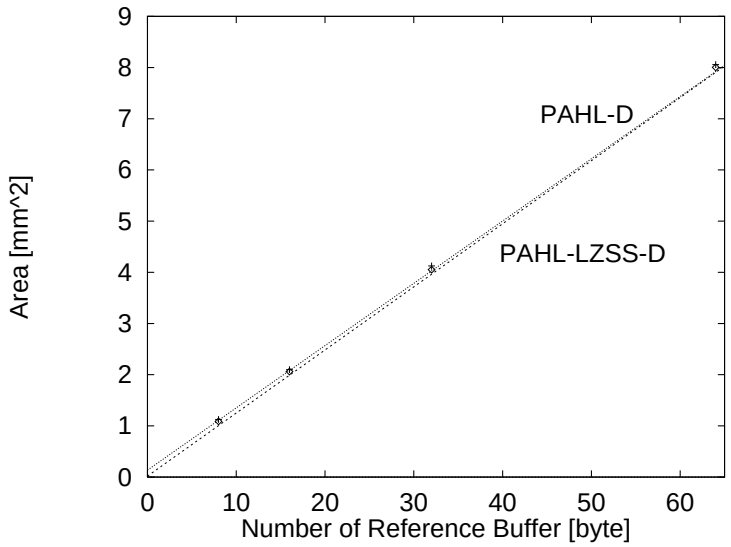


図 4.8: PAHL-LZSS-D の実装面積

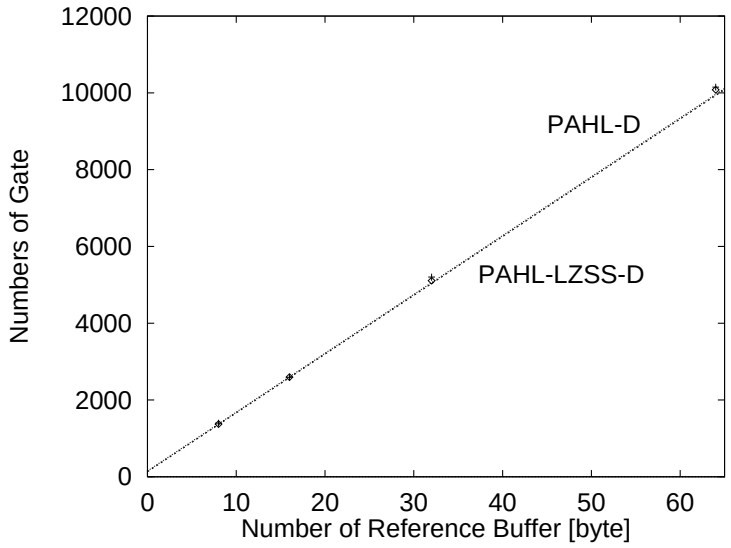


図 4.9: PAHL-LZSS-D のゲート数

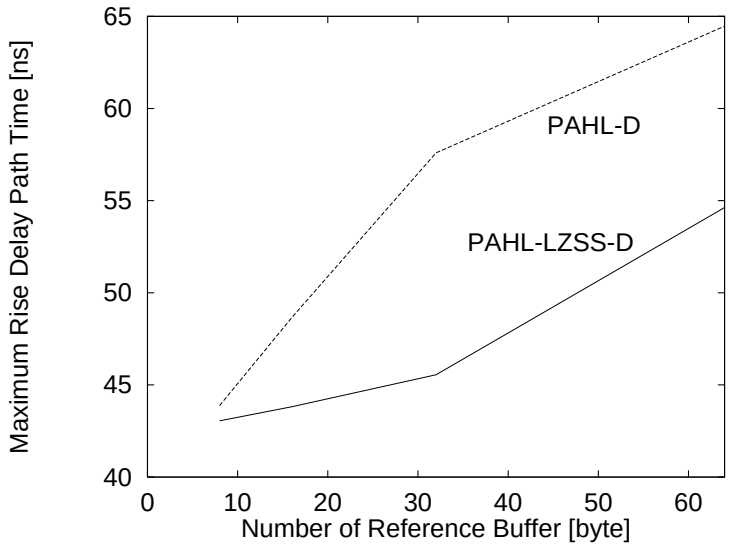


図 4.10: PAHL-LZSS-D の最大遅延時間

表 4.2: PAHL-LZSS-D の予測性能

参照部バッファ長 [byte]	1K	4K	8K
消費電力 [W/MHz]	0.5	2.0	4.0
実装面積 [mm^2]	125	500	1000
ゲート数	160K	640K	1280K
最大遅延時間 [ns]	85	95	100
スルー・レート [Mbyte/sec]	11.8	10.5	10.0

4.5 高速 LZSS 符号化・復号化並列処理アーキテクチャ - PAHL-LZSS

4.5.1 PAHL-LZSS の基本構成

PAHL-LZSS は、PAHL-C、PAHL-D を元にして PAHL を構成するのと同様に、PAHL-LZSS-C、PAHL-LZSS-D を元にして構成している。

PAHL-LZSS-C は PAHL-C への若干の機能の追加で実現され、また PAHL-LZSS-D と PAHL-D とは基本的に回路構成は同じである。また、PAHL は PAHL-C へ PAHL-D 固有の機能を追加することにより容易に実現される。以上により、PAHL-LZSS は、PAHL-LZSS-C へ PAHL-LZSS-D 固有の機能を追加することにより、または PAHL へ PAHL-LZSS-C 固有の機能を追加することにより、容易に実現できる。

次に PAHL-LZSS の基本構成と動作アルゴリズムを示す。

動作アルゴリズムは、

(1) 符号化 (PAHL-LZSS-C と同じ)

(2) 符号化 / 復号化切替

(3) 復号化 (PAHL-LZSS-D と同じ)

となる。(1)、(3) はそれぞれ PAHL-LZSS-C、PAHL-LZSS-D と同様の処理アルゴリズムである。(2) における符号化 / 復号化の切替では、PAHL-LZSS に対して切替を要求する信号を出し、その後 PAHL-LZSS 側から切替完了の信号が返された後、次の処理を実行するアルゴリズムとなる。切替信号は、符号化記号系列・復号化される符号が全て入力された後に入力されものとする。具体的には、

(a) 符号化 → 復号化切替

  (i) 符号化 / 復号化切替信号入力

  (ii) cell2 部内符号データ出力

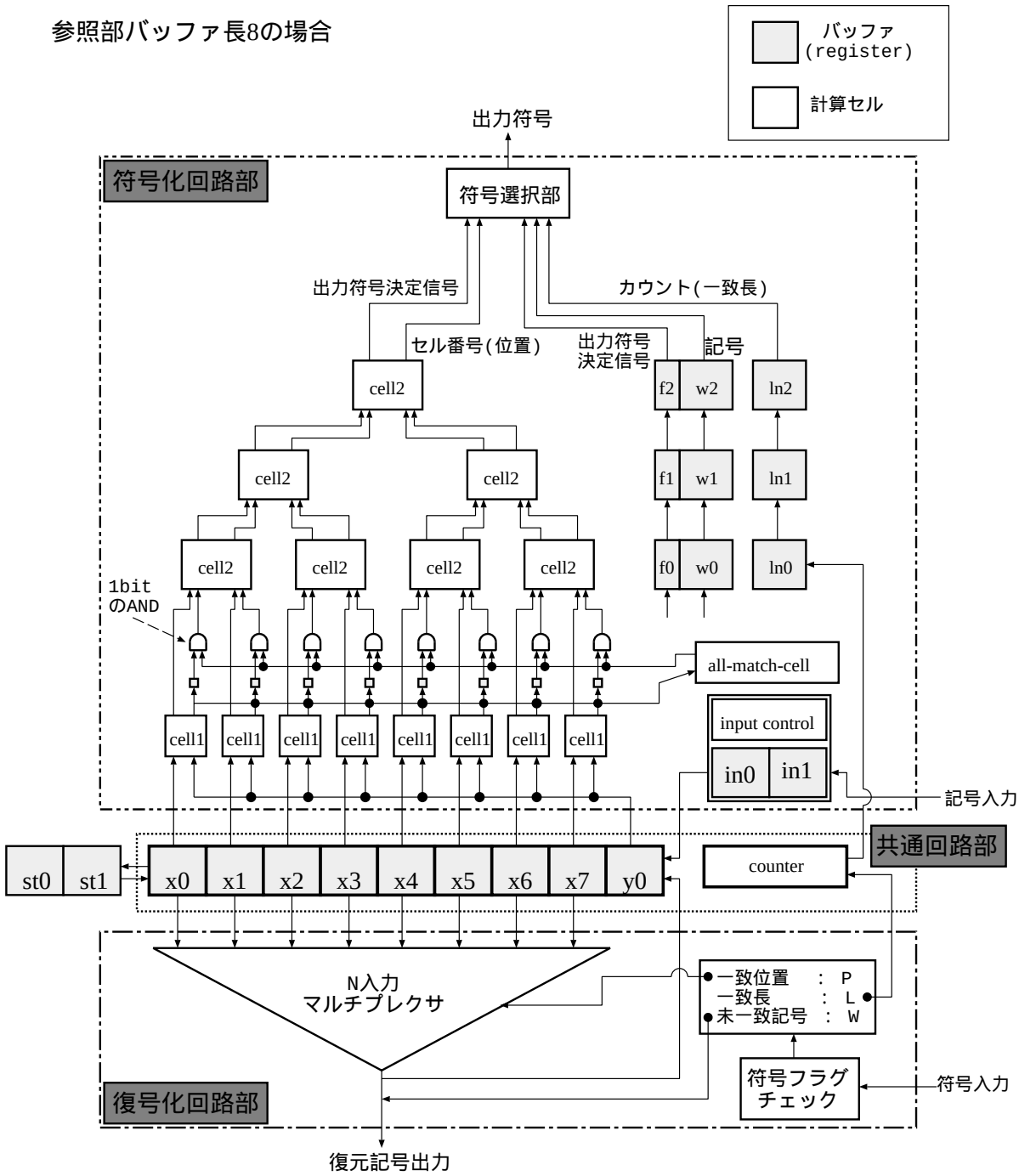


図 4.11: PAHL-LZSS の基本構成

cell2 部及び一致位置、一致長遅延用バッファ部を段数分 ($\lceil \log_2 N \rceil$ 回) 動作させ、cell2 部内のデータを出力する。出力されたデータの中に符号に該当するデータがあった場合、それは出力符号とみなされる。

(iii) 復号化処理のための各初期化
(iv) 復号化開始

(b) 復号化 → 符号化切替

(i) 符号化 / 復号化切替信号入力
(ii) 符号化処理のための各初期化
(iii) 符号化開始

となる。復号化から符号化への切替は容易に実装可能であるが、符号化から復号化への切替時には、切替信号入力直前までの cell1 部での結果とカウント値、符号化部バッファの先頭位置の記号 (cell2 部に送られるデータ) を出力する必要が生じる。このため、符号化から復号化処理が実行される状態にする間に、cell2 部内にまだ残っているデータを出力させる必要があり、切替信号入力後に cell2 部を $\lceil \log_2 N \rceil$ 回実行させなければならなくなる。この部分が符号化 → 復号化において要する計算となる。

PAHL-LZSS の動作アルゴリズムが、符号化・復号化においては PAHL-C、PAHL-D と同様であるので、計算量は以下のようになる。

(a) 符号化 ： 一符号あたり L + 2 ステップ → O(L)
(b) 復号化 ： 一符号あたり L + 2 ステップ → O(L)
(c) 符号化 → 復号化切替 ： $\lceil \log_2 N \rceil$ ステップ → O($\lceil \log_2 N \rceil$)
(d) 復号化 → 符号化切替 ： 1 ステップ → O(1)

4.5.2　PAHL-LZSS の性能評価

PAHL-LZSS の性能評価についても PARTHENON を使用して、SFL による論理設計 (動作記述)、設計した回路の論理検証 (動作検証)、及び論理合成を行ない、実現される回路規模、動作速度を求めた。なお、論理合成に用いたデータは PAHL 等の場合と同様、DEMO 社の demo ライブラリ (1.5 μ CMOS テクノロジ) を使用した。

PAHL-LZSS の設計においても、PAHL と同様、符号化 / 復号化両方の設計 (動作記述) が必要になる。それぞれは PAHL-LZSS-C / PAHL-LZSS-D と同様に設計されるものになるが、スライド窓バッファとカウンタは共用になる。従って PAHL-LZSS でも、PAHL-LZSS-C に対し、アーキテクチャの構成要素として新規に位置決定部 (N 入力マルチプレクサ) と符号化 / 復号化切替時処理部の二つが追加される。

[1] PAHL-LZSS の論理合成結果

論理合成結果から得られた PAHL-LZSS の消費電力、実装面積、ゲート数を PAHL-C、PAHL-D の場合と共に図 4.12、4.13、4.14に示す。

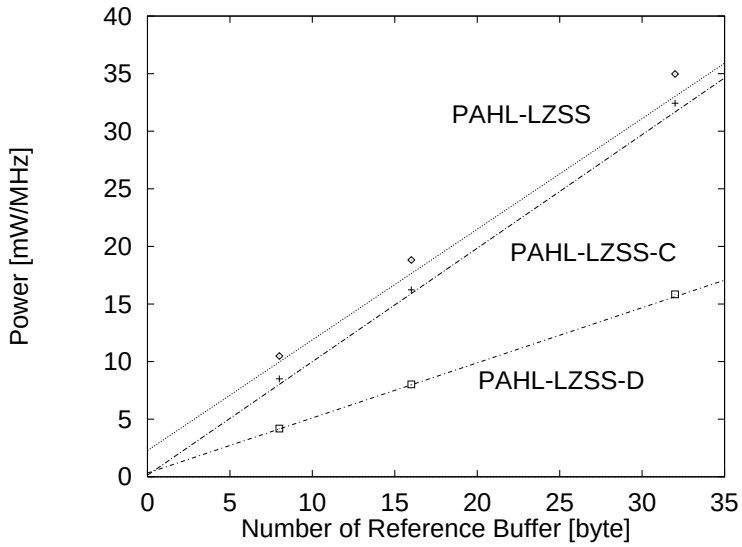


図 4.12: PAHL-LZSS、PAHL-LZSS-C、PAHL-LZSS-D の消費電力

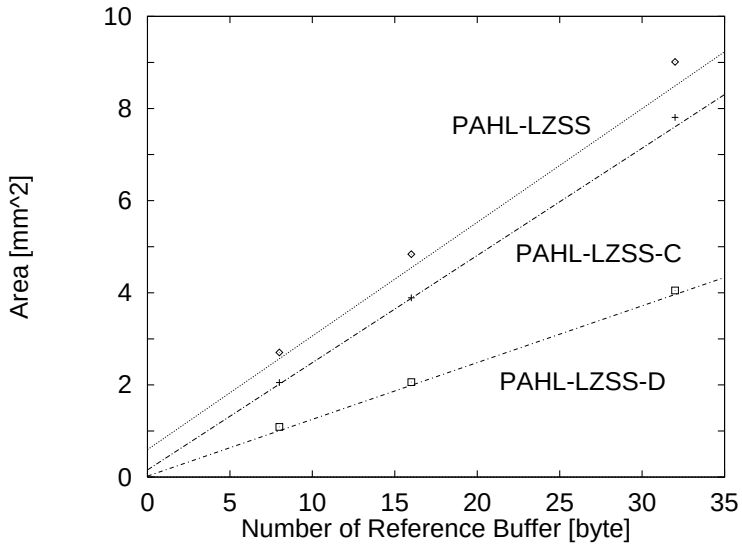


図 4.13: PAHL-LZSS、PAHL-LZSS-C、PAHL-LZSS-D の実装面積

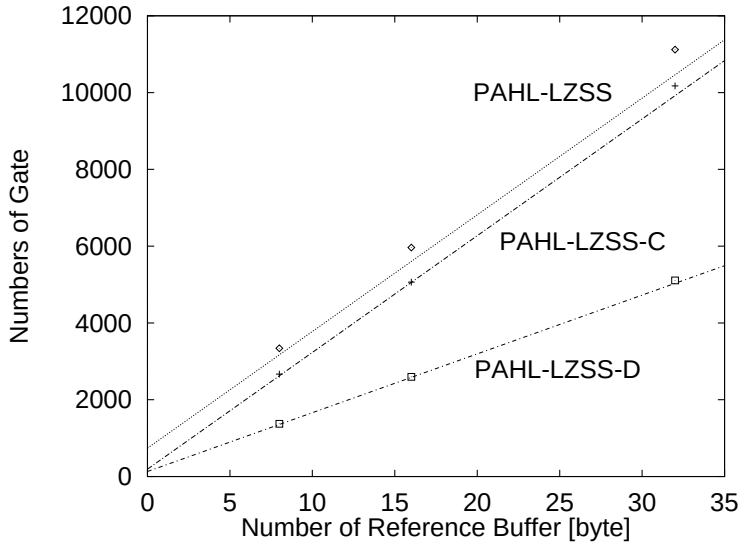


図 4.14: PAHL-LZSS、PAHL-LZSS-C、PAHL-LZSS-D のゲート数

論理合成によって得られた PAHL-LZSS の消費電力、実装面積、ゲート数は、PAHL-LZSS-C のものと同様の結果が得られた。つまり、各々の傾向は PAHL、PAHL-C、PAH-D での関係とほぼ同様であり、回路規模としては、符号化部のみの場合 (PAHL-LZSS-C) からはほとんど増加していないことが確認された。

遅延時間についても、符号化・復号化処理それぞれについては PAHL、PAHL-C、PAHL-D と同じ計算セルを用いており、アーキテクチャ自体もほぼ同じである (符号化 / 復号化切替信号の扱いに伴い、入力信号を扱う部分に若干の変更 (追加) がある)。構成している機能、計算セルも PAHL-C、PAHL-D と同じものを用いているので、処理時間 (遅延時間) については、PAHL-C、PAHL-D に関しての議論がそのまま適用している。

[2] PAHL-LZSS の予測性能

[1] で得られた論理合成による結果から、N = 1K の場合についての PAHL-LZSS の予測性能を、PAHL-LZSS-C、PAHL-LZSS-D と共に表 4.3 に示す。

予測性能より、PAHL-LZSS の場合も、PAHL の場合と同様に符号化 / 復号化を統合することにより、より少ない回路規模により符号化・復号化両方を実現することが可能となる。

表 4.3: 予測される PAHL-LZSS の性能及び PAHL-LZSS-C、PAHL-LZSS-D との比較

	PAHL-LZSS	PAHL-LZSS-C	PAHL-LZSS-D
消費電力 [W/MHz]	1	1	0.5
実装面積 [mm^2]	270	240	125
ゲート数	330K	300K	160K
遅延時間 [ns]	85	50	85
レート [Mbyte/sec]	11.8	20	11.8

第 5 章

モジュール化による PAHL の実装

5.1 PAHL、PAHL-LZSS の実装における問題

PAHL(及び PAHL-LZSS) の論理合成において得られた予測性能から、実装するにあたり最も重要、もしくは問題となる点として、その回路規模が挙げられる。ASIC として実現するためには、できるだけ回路規模の小さいアーキテクチャであることが望ましい。しかし、実用上のパラメータ値 (N = 1K ~、M = 32 ~ 256) における PAHL、PAHL-LZSS については、ゲート数が N = 4K 程度で約 100 万ゲート程度と、かなり膨大な値になることが予測されている。

もし、これをワンチップとして実現しようと考えた場合、現在の VLSI/ULSI 技術では不可能ではないが、本来の ASIC としての実現の容易性や、チップの信頼性などから考えると、できるだけ小さいチップとして実装することが望ましい。

ここで、PAHL(及び PAHL-LZSS) において、回路全体のゲート数の約半数を占めているのが、スライド窓バッファ(参照部バッファ(辞書バッファに相当する部分)) であることを確認している。従って、1 チップあたりの回路規模の小さいアーキテクチャを実現するためには

(i) 参照部バッファを外部メモリとして実現

(ii) 参照部バッファを内部 RAM として実現

(iii) PAHL の各部を機能別に分割し、それらをある程度まとめて一つのチップ (モジュール) として実装する。

というアプローチが考えられる。しかし、(i) のように外部メモリとして実現した場合、PAHL 自身の回路規模 (消費電力、実装面積、ゲート数) としては約半分に削減することができるが、一致長探索部 (cell1 部) と参照部バッファ間の、並列に実行されているデータ転送部での転送速度が著しく低下するものと考えられる。また (ii) の内部 RAM としての実装についても、(i) に比べて速度的な低下はある程度抑えられるものと考えられるが、回路

規模としてはそれほど効果が上がらないものと予測される。いずれにしても、実用上の N の値においては、(i)、(ii) の場合それぞれの手法により実装したとしても、大規模な回路となってしまうものと予測される。

そこで、PAHL をより容易に実装するための手法の一つとして、(iii) で示したような、PAHL を分割し、それらをある程度まとめて一つのモジュールとして実現するアプローチを提案する。

5.2 PAHL-C の実装 ~ モジュール化 ~

PAHL、PAHL-LZSS は、双方とも元々は PAHL-C のアーキテクチャが基本となっており、アーキテクチャの構成もほぼ同様である。そこで以下では、PAHL-C の分割について述べる。PAHL-C における分割法、及びその結果は、そのまま PAHL、PAHL-LZSS へ応用できる。

5.2.1 モジュール化のための分割法

PAHL-C は主にスライド窓バッファ(辞書バッファ)、一致長探索部 (cell1 部)、一致長決定部 (cell2 部)、一致判定終了決定セル (all-match-cell)、カウンタ、各種データ用バッファ (レジスタ) から構成されている。PAHL-C を分割するにあたっては、以下のことを考慮して分割する必要がある。

(1) 各セル、各機能間のデータ依存

(2) 構成するにあたっての回路規模

(3) 入出力数

(1) では、動作速度面を考慮する場合、各モジュール間でできるだけ相互のデータ依存がなく、それぞれが並列に (独立に) 動作されるように分割することが重要である。(2) では、例えばある機能を持つモジュールの回路規模が突出して大きくなることや、所望の PAHL-C を構成した場合の、各モジュールの数、大きさができるだけ均一になることが望ましい。(3) では、(2) との兼ね合いもあるが、ビットパラレルなデータ入出力を想定すると、入出力ビット幅を考慮する必要がある。

5.2.2 各機能モジュール

前節で示した (1)、(2)、(3)、特に (1) を考慮して、次に示す三種類のモジュールへの分割を提案する。

[1] 一致長探索モジュール

一致長探索モジュールは、図 5.1のように、cell1 部と参照部バッファにより構成されている。このモジュールの 1 モジュールあたりの回路規模については、前節 (2)(3) を考慮して、参照部バッファ長 128[byte]、cell1 数 128 個と決定した。次の、この場合での 1 モジュールあたりの性能を、PAHL 等の場合と同様に PARTHENON による論理合成結果として表 5.1に示す。

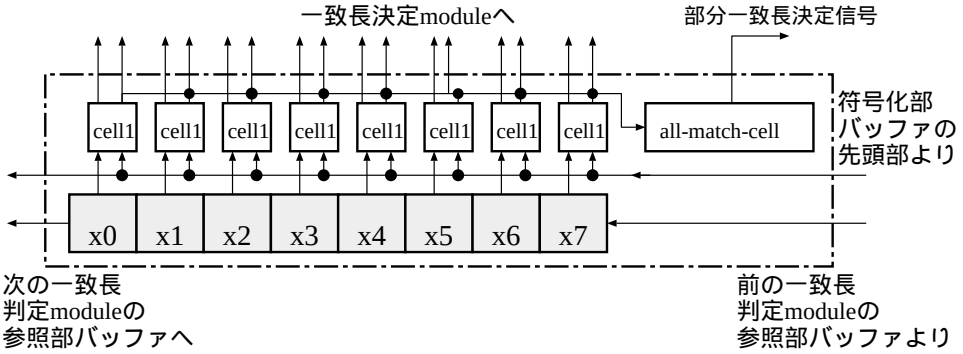


図 5.1: 一致長探索モジュール

表 5.1: 一致長探索モジュールの性能 (ref. buffer : 128, cell1 : 128)

消費電力 [mW/MHz]	57.6
実装面積 $[mm^2]$	14.7
ゲート数	19876
最大遅延時間 [ns]	55.95

[2] 最長一致選択モジュール

最長一致選択モジュールは、図 5.2に示すように cell2 部によって構成されている。cell2 部は、完全二分木状に構成されるものを実現しなければならないため、一種類の最長一致選択モジュールで、PAHL-C 全体の cell2 部を実現するのは、構成可能の木の規模 (cell2 数) の条件により困難 (大きな制限が生じる) である。そのため一致長決定モジュールでは、1 モジュールあたりの回路規模や、全体のモジュール数を考慮した場合、回路規模に応じて同機能を持つ数種のモジュールが必要となる。

表 5.2に、1 モジュールあたりの cell2 の数 127 個の場合の性能を示す。

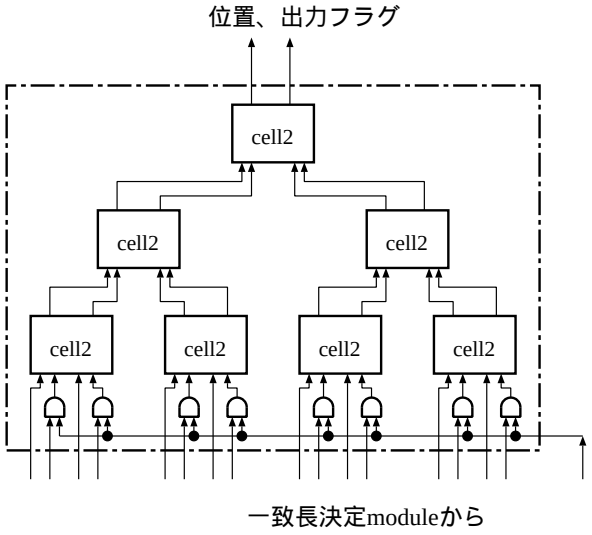


図 5.2: 最長一致選択モジュール

表 5.2: 最長一致選択モジュールの性能 (cell2 : 127)

消費電力 [mW/MHz]	22.3
実装面積 $[mm^2]$	6.0
ゲート数	7679
最大遅延時間 [ns]	34.5

[3] カウンタモジュール

カウンタモジュールは、図 5.3のように counter 及び各種バッファ類カウンタ module により実現される。ここで counter のビット幅と各種データ用バッファが $\lceil \log_2 N \rceil$ に比例し増加する程度である。カウンタのビット幅 8bit、N=4096 の場合、論理合成の結果でも 3000 ゲート程度であり、また入出力ビット長から考えても、カウント module については実用上のバッファ長において、一つの module として容易に実現できるものと思われる。表 5.3にカウンタのビット幅 8[bit] の場合の性能を示す。

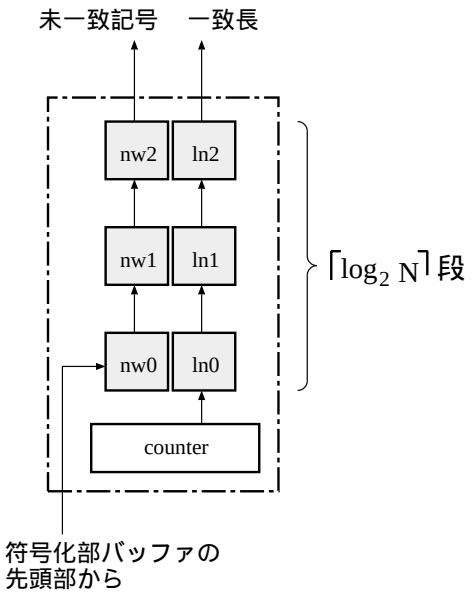


図 5.3: カウンタモジュール

表 5.3: カウント module の性能

参照部バッファ長	1K	4K
消費電力 [mW/MHz]	5.03	7.34
実装面積 [mm^2]	1.39	2.02
ゲート数	1824	2660
最大遅延時間 [ns]	29.69	29.81

5.3　各機能モジュールによる PAHL-C の構成

各機能モジュールを組み合せることにより、所望の回路規模の PAHL-C を構成することが可能である。図 5.4、表 5.4に, 参照部バッファ N = 1K[byte] の場合の PAHL-C のモジュール構成を示す。

図 5.4のように、三種類の機能別モジュールにより、参照部バッファ長 1K[byte] の PAHL-C を構成する場合、表に示したモジュールによ構成することができる。

ここで問題となるのが、最長一致選択モジュール (cell2 部) として複数種 (cell2 数が異なるもの) が必要となることである。理想的には一種類の最長一致選択モジュールにより構成されることが望ましい。しかし、ある完全二分木を一種類の小さな二分木で構成することは、ノード数などで相当の制限を受けることになる。これを解決するための方法としては、最長一致選択モジュールに対し、その中の任意の cell2 部 (任意の sub tree) を利用することができるように、最長一致選択モジュール内で動作させる部分を外から制御できるようにする方法が考えられる。但し、全ての最長一致選択モジュールに制御機能が付加されることにより、1 モジュールあたりの回路規模の増加、動作速度の低下が予想される。

カウンタモジュールについても、モジュール内の構成がパラメータによって異なるものとなるが、論理合成結果などから得られた回路規模を考えると、カウンタモジュールを構成している部分は、無理にモジュール化する必要はないものと判断できる。

以上に示した PAHL-C の機能別の分割及びモジュール化により、必要なモジュールを組み合せることにより、所望のパラメータ値 (N、M) での PAHL-C を構成することが可能である。同時に PAHL、PAHL-LZSS は回路構成が PAHL-C を基本としていることより、PAHL、PAHL-LZSS についても同様にモジュールにより構成することができる。

表 5.4: module 化による PAHL-C の構成 (ref. buffer : 1K[byte])

	1module の内部セル数	module 数
一致長探索 module	バッファ長 : 128[byte] cell1 : 128	8
カウント module	カウンタ bit 幅 : 8[bit]	1
一致長決定 module	cell2 : 127 cell2 : 7	8 1

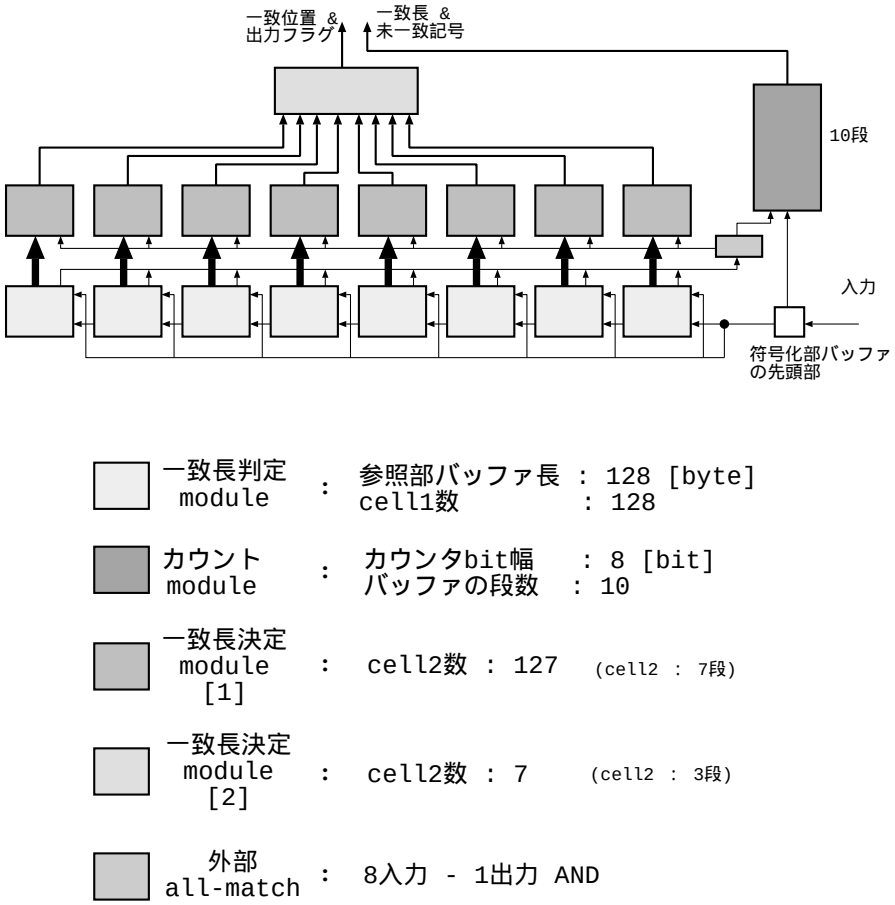


図 5.4: PAHL-C の分割モジュールによる構成 (N = 1K[byte])

第 6 章

結論

6.1 はじめに

コンピュータ技術の発達とともに、コンピュータが我々の日常生活の中に広く普及している現在では、各種の情報・データが様々な要求により利用されている。また Internet 等をはじめとするの各種 Network などでは、様々なデータ伝送が頻繁に行なわれ、それらのデータを運用するための効率的な蓄積・管理が求められている。以上の状況におけるデータの取り扱いにおいての重要点の一つが、データのより高速な通信 (伝送)、また効率的な蓄積・管理である。このことは、リアルタイムなデータ処理、効率的なデータ運用に大きな役割をはたすこととなり、我々の行動、生活などの様々な部分に影響を与える。

より効率的なデータ伝送、蓄積への重要な技術が、データ圧縮 (符号化) 技術である。データを圧縮することにより、有限な蓄積資源をより有効に活用し、データ伝送時間の短縮、単位時間当りのデータ伝送量の増加を可能とする。

本論文では、高速なデータ圧縮アーキテクチャの構築を目的とし、その一つとして高速 LZ77 符号化・復号化並列処理アーキテクチャ - PAHL を提案している。データ圧縮ハードウェアを実現することにより、

- より高速なデータ圧縮 (符号化)
- 他のデータ処理との並行 (独立) 動作

が可能となり、より効率的なデータ運用が実現される。

本論文で提案している PAHL は、主なデータ圧縮法の一つである LZ77 符号 (LZSS 符号) を高速に実行する並列処理アーキテクチャである。LZ77 符号は

- 様々な形式の情報源に対し有効である。
- 符号化・復号化処理が比較的単純である。

○　現在あるいろいろなデータ圧縮アプリケーションに利用されている。

などの特徴を持つデータ圧縮法である。以上の特徴は、データ圧縮ハードウェアを実現するうえでの重要な条件に沿うものである。

6.2　高速 LZ77 符号化・復号化並列処理アーキテクチャ - PAHL

本研究では、まず始めに、高速な LZ77 符号化を実現する、高速 LZ77 符号化並列処理アーキテクチャ - PAHL-C を提案した。PAHL-C において、より高速な LZ77 符号化を実現するための重要な部分となるのが、LZ77 符号における生成符号の統計的特徴を積極的に生かした点である。生成符号の統計的特徴とは、各々の符号化過程で得られる最長一致記号系列長 L の出現頻度の統計的特徴のことである。

まず、統計的特徴を積極的に生かすことができる並列アルゴリズムとして、

- 参照部バッファ(バッファ長 N) 全位置での一致記号系列探索 (比較演算) 並列に実行。
- 全位置での一致記号系列探索が完了した時点で、一符号あたりの一致記号系列探索を終了。
- 得られた一致記号系列から最長一致記号系列 (記号系列長 L) を選択。

を実現するものを提案した。この並列アルゴリズムにより、$L + 1$ 回の比較演算での最長一致記号系列探索を実現することが可能である。

また、LZ77 符号においての実際の統計的特徴の調査により

- 生成符号の持つ L の大部分が、N に比べて十分に小さい (L のほとんどは $0 \sim 20$ 程度)
- 一符号あたりの平均最長一致記号系列長も最大でも 8 程度

という結果が得られた。この結果から、一符号あたりの最長一致記号系列探索に要する比較演算を $L + 1$ 回にすることにより、計算量を大幅に削減 (O(N) から L 回程度へ) することが可能であること確認された。以上の提案手法を実現するアーキテクチャとして、PAHL-C では、並列処理を応用することにより一符号あたりの計算回数を $L + 2$ 回にすることを実現している。

本研究では、提案した PAHL-C の性能評価に、ハードウェア設計･支援システム PARTHENON を使用した。PARTHENON 上のハードウェア記述言語である SFL により動作記述、及び PARTHENON 上のシミュレータによる動作検証を行ない、期待通りの動作をするアーキテクチャを設計した。その後論理合成により、実装された場合の予測性能を求めた。その

結果、実用上のパラメータ値とされている参照部バッファ長 N = 4K[byte]、符号化部バッファ長 M = 32[byte] の場合において、ゲート数として約 100 万ゲート、最大遅延時間 (動作速度) として約 50[ns](1.5 μ CMOS テクノロジデータを使用) という結果が得られ、この結果により、(本使用テクノロジデータにおいて) 約 20[Mbyte/sec] での符号化が可能となることが確認された。

次に、高速な LZ77 復号化を実行する、高速 LZ77 復号化並列処理アーキテクチャ- PAHL-D を提案した。LZ77 復号化は、符号から得られた一致位置 P、一致長 L、未一致記号 W により、参照部バッファ上の P から始まる長さ L の記号系列と未一致記号 W を復元記号系列として出力する、非常に簡単な処理となる。

PAHL-D は、参照部バッファ、参照部バッファの所望位置の記号を選択するための N 入力マルチプレクサ、カウンタから成る、非常に簡素な構成である。PAHL-D では、一符号あたりの (長さ L+1 の) 復元記号系列を得るのに、L + 2 回の計算回数を要する。つまり、PAHL-C と同様に O(L) での処理となる。PAHL-D についても、PAHL-C の場合と同様に、PARTHENON による設計、動作検証、論理合成を行ない、同テクノロジデータ、N、M において、ゲート数約 60 万ゲート、最大遅延時間約 95[ns]、復号化スルー・レート約 10[Mbyte/sec] という予測性能が確認された。

一般的にデータ圧縮 (符号化・復号化) を利用する場合、符号化 / 復号化のいずれか一方のみが実行できればよいという状況はまれであり、普通は符号化・復号化の双方が実行できなければならない。そのためには符号化・復号化ハードウェアの両方が実装されていなければならない。しかし実装面積、コストの点から見ると好ましいことではない。LZ77 符号では、符号化 / 復号化が対応する処理であり、言いかえると、同様の処理が含まれていることを意味する。従って、ハードウェア化する場合、符号化 / 復号化がそれぞれ同様の機能部を持つものと考えられる。

PAHL-C、PAHL-D には、各々に共通の機能部として

- 参照部バッファ(辞書バッファ)
- カウンタ

を持つ。特に参照部バッファは回路規模に大きく依存する部分である。このことから、共通の処理を実行する部分を共有することで、回路規模の増加を押さえ且つ PAHL-C と PAHL-D を統合できる。そこで、PAHL-C、PAHL-D を効率的に統合した、高速 LZ77 符号化・復号化並列処理アーキテクチャ - PAHL を提案した。PAHL を構成する機能 (計算セル) は PAHL-C、PAHL-D で用いられているものと同じであり、PAHL の基本構成は PAHL-C と同じである。

PAHL についても、PARTHENON により設計、動作検証、論理合成を行ない、予測性能を得た。その結果、回路規模としては PAHL-C に比べて多少の増加はあるが、PAHL-C、PAHL-D 別個に実装した場合に比べ十分小さいものであることを確認した。また動作速度に関しても、PAHL-C、PAHL-D とほぼ同様の結果が得られた。以上のことから、LZ77 符

号を実現したハードウェアである PAHL-C、PAHL-D を効率的に統合可能であり、統合が十分有効であることが確認された。

6.3 高速 LZSS 符号化・復号化並列処理アーキテクチャ - PAHL-LZSS

現在、LZ77 符号として一般で利用されている符号は、その多くは、LZ77 符号に関し符号の持つ冗長性をより削減した、LZSS 符号と呼ばれるものである。従って、LZSS 符号を実行可能なハードウェアを構成することが、今後の応用分野でのデータ圧縮ハードウェアの利用に際しては重要となる。LZSS 符号は

- 基本的な処理は LZSS と同じ
- 出力される符号の形式を複数にすることで、符号の持つ冗長性を削減

という特徴を持つ。従って、符号化・復号化における (計算) 処理自体は LZ77 符号と同じであるので、PAHL を容易に LZSS 符号へ拡張することが可能であるものと考えられる。そこで次に、PAHL を LZSS 符号へ拡張した、高速 LZSS 符号化・復号化並列処理アーキテクチャ - PAHL-LZSS を提案した。

LZSS 符号化を実現した高速 LZSS 符号化並列処理アーキテクチャ - PAHL-LZSS-C は、使用計算セル、基本構成は PAHL-C とほぼ同じである。生成中間符号の持つ最長一致記号系列長 L により、記号符号が出力されるかポインタ符号が出力されるかが決定される。このとき、得られた L と出力される符号形式の関係によって、一度一致長探索部を通過した記号 (スライド窓バッファからはずれた記号) が、次の符号化に用いられる場合が生じる。この部分を実装するために、一時的に記号をストックしておく機能と、参照部バッファ内記号系列を反対にシフトする機能が必要となる。以上二つの機能を追加し、若干の動作アルゴリズムの変更によって、PAHL-C から容易に PAHL-LZSS-C への拡張 (変更) を行なうことが可能である。

PAHL-LZSS-C についても、PAHL-C などと同様の条件により、PARTHENON を用い、設計、動作検証、論理合成を行ない、予測性能を得た。その結果、同条件、同パラメータ (N、M) 値における予測性能は、回路規模は PAHL-C にくらべ多少増加したものの、動作速度ともに、PAHL-C の場合とほぼ同様の結果であることが確認された。

LZSS 復号化では、符号の持つフラグにより符号形式を判定し、それが記号符号の場合、符号内の記号を復元記号として出力し参照部バッファの更新を実行し、ポインタ符号であれば LZ77 復号化と同様の処理を実行する。従って、PAHL-LZSS-D の場合も、PAHL-D の構成をほぼそのまま応用することができる (PAHL-D の機能に、フラグによる符号判定、及び記号符号時の記号出力部を追加するのみである)。以上にように、PAHL-D から PAHL-LZSS-D へ容易に拡張された。PAHL-LZSS-D について得られた予測性能は、PAHL-D とほぼ同じ結果であった。

PAHL-LZSS-C、PAHL-LZSS-D それぞれについても、構成している機能が、それぞれ PAHL-C、PAHL-D と同様であるので、PAHL の場合と同じプロセスにより統合することが可能である。実際に構成した PAHL-LZSS についても、PARTHENON を用い、設計、動作検証、論理合成を行ない、得られた予測性能は、回路規模としては PAHL よりも多少増加したものの、その他の性能としては PAHL とほぼ同じ性能であることを確認した。

以上にことから、LZ77 符号を実現したアーキテクチャである PAHL から、LZ77 符号の改良手法である LZSS 符号を実現するアーキテクチャである PAHL-LZSS へ、容易に拡張可能であり、性能についても PAHL とほぼ同じものとなることが確認された。このことは、LZ77 符号の他の改良手法 (LZB、LZR など) への拡張も容易に実現できることを示しているものと見ることができる。

6.4　PAHL の実装 ~ モジュール化 ~

本論文では、LZ77 符号及び LZSS 符号を高速に実行することができるアーキテクチャである PAHL、PAHL-LZSS を提案し、ハードウェア設計・支援システム PARTHENON を用い、実装した場合の予測性能を求めた。ここで問題として挙げられたことが、実用上のパラメータ値 (参照部バッファ長 $N = 1K[byte]$ ~、符号化部バッファ $M = 32$ ~ $256[byte]$) の条件での (大きな) 回路規模であった。実用上の N、M でのアーキテクチャをワンチップに実装することは、不可能ではないが、ASIC としての実現容易性やその安定性から実現性に多少欠けたものであると考えた。そこで、それを解決するための一つの手法として、PAHL(及び PAHL-LZSS) を機能別に数種類のモジュールに分割し (そのモジュールをワンチップとして)、それらのモジュールを組み合せることで、所望の回路規模の PAHL を実装する手法を提案し、どのように分割、モジュール化しているかを示した。

6.5　おわりに

本論文では、LZ77 符号、及び LZSS 符号を高速に実行する並列処理アーキテクチャとして PAHL、PAHL-LZSS を提案し、その予測性能を求めた。本提案アーキテクチャである PAHL、PAHL-LZSS は、既存の LZ77 符号化アーキテクチャと比較して、符号化に要する計算量を大幅に削減することが可能なり、それにより単位時間あたりに符号化される情報量を大幅に増加することが可能である。しかし、今回予測性能を求めるために使用したテクノロジデータ (1.5 μ CMOS データ) は、現在の VLSI/ULSI で用いられているものと比較して、かなり性能の低いデータである。よって、現在使用されているレベルのテクノロジ (0.35 μ CMOS 程度) では、本論文で示した性能よりも現状に合った、良い性能を得ることができるものと考えられる。そこで今後の課題として、まず PAHL、PAHL-LZSS をハードウェアとして実装し、実際の動作性能を求めることが挙げられる。これにより、実装した場合の性能や、現実に実装した場合の問題が明らかになるものと考えられる。

また、提案した PAHL、PAHL-LZSS を、LZ77 符号を応用しているものとして現在利用されているデータ圧縮アプリケーション (LHA、gzip など) に応用、拡張することが挙げられる。ただし、現在 LZ77 符号が応用されているデータ圧縮アプリケーションでは、LZ77 符号 (または LZSS 符号) のみを利用しているわけではなく、Huffman 符号や算術符号などと組み合せた形で (ハイブリッド形式で) 利用されていることが多い。そのため、LZ77 符号 (LZSS 符号) とは別の符号化についても、実装する必要がある。

謝辞

本研究を行なうにあたり、全般的な御指導、御助言を頂き、ならびに多大なご支援、御励ましを頂いた東北大学工学部 阿曽弘教授に心から感謝致します。

本研究をまとめるに当って，適切な御助言をいただいた，東北大学工学部坪内和夫教授，川又政征教授に心から感謝致します．

東北大学工学部通信工学科 大町真一郎助手、黒岩丈介助手、東北大学情報処理教育センター 後藤英昭助手、山口大学経済学部 成富 敬助教授には、日頃から研究において多くの適切な御意見、御助言を頂き、同時に熱心な御討論、御協力を頂きました。ここに深く感謝致します。

本研究を進めるにあたり、ハードウェア設計・支援システム PARTHENON を貸し出して下さいました NTT 、および PARTHENON 研究会の皆様に感謝致します。

最後に、日頃から熱心な御討論、御協力を頂きました阿曽研究室の皆様に深く感謝致します。

参考文献

[1] N. Ranganathan and Selwyn Henriques, "High-Speed VLSI Designs for Lempel-Ziv-Based Data Compress ion", IEEE Trans. Circuit & Syst., vol.40, no.2, pp.96-106, 1993

[2] R. Zito-Wolf, "A Broadcast/Reduce Architecture for High-Speed Data Compression", in Proc. IEEE I nt. Symp, on Parallel and Distributed Processing, Dallas, TX, Dec. 1990

[3] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression", IEEE Trans. Infor m. Theory, vol. IT-23, pp. 337-343, 1977

[4] James A. Storer and John H. Reif, "A Parallel Architecture for High-Speed Data Compression", Journal of Parallel and Distributed Computing 13, pp. 222-227, 1991

[5] Amar Mukherjee and N. Ranganathan, "MARVEL: A VLSI Chip for Data Compression Using Tree-Based Codes", IEEE Trans. VLSI Sys., vol. 1, no.2, June 1993

[6] Amar Mukherjee, N. Ranganathan, M. Bassiouni, "Efficient VLSI Design for Data Transformation of Tree-Based Codes", IEEE Trans. Circuit & Syst., vol.38, no.3, 1991

[7] J.Jiang, S.Jones, "Parallel design of arithmetic coding", IEE Proc. Comput. Digit. Tech., vol.14 1, no.6, 1994

[8] 花田孝郎 "文字列のパターンマッチ法," 情報処理, vol.24, no.4, pp494-498, Apr. 1983.

[9] 小川隆一, 菊地芳秀, 高橋恒介, "フルテキスト・データベースの技術動向," 情報処理, vol.33, no.4, pp404-412, Apr, 1992

[10] 高橋恒介, 永井肇, 山田八郎, 平田雅規, "ストリング・マッチング・ハードウェアのアーキテクチャ,"

[11] 宮原末治，近藤利夫，多田俊吉，"SIMD 型並列プロセッサを用いたフルテキスト検索," 情報処理学会論文誌，vol.33, no.3, pp397-404, Mar. 1992. 信学技法, CPSY86-57, pp.57-68, 1986.

[12] 山本博資, "ユニバーサルデータ圧縮アルゴリズム：原理と手法," 情報処理, vol.35, no.7, pp.600-608, July 1994.

[13] 植松友彦, "LZ77 符号, " 文書データ圧縮アルゴリズム, pp134–141, ＣＱ出版株式会社, 東京都, 1994.

[14] 梅尾博司，"シストリック・アレイ，" 情報処理，vol.30, no.1, pp.15-28, Jan. 1989.

[15] 飯国洋二，酒井英昭，得丸英勝，"1 次元シストリックアレイの設計法，" 電子情報通信学会論文誌 D，vol.J-71-D，no.8，pp.1480-1486, Aug. 1988.

研究業績

○ 発表論文

- 藤岡豊太, 阿曽弘具, "高速 Lempel-Ziv 符号化 / 復号化並列処理アーキテクチャ, " 電子情報通信学会論文誌 (D-I) (投稿中)

○ 学会発表

- 藤岡豊太, 阿曽弘具, "高速 Lempel-Ziv 符合化ハードウェアアーキテクチャの構築, " 1995 年電子情報通信学会ソサイエティ大会, D-38, 1995.
- 藤岡豊太, 阿曽弘具, "高速 Lempel-Ziv 符号化用並列処理アーキテクチャ, " 信学技報, vol.95, no.244, CAS95-64, pp.37-43, Sept. 1995.
- 藤岡 豊太, 阿曽 弘具, "PARTHENON による高速 Lempel-Ziv 符号化アーキテクチャの設計", 第 7 回パルテノン研究会, pp.19-26, 1995.
- 藤岡豊太, 阿曽弘具, "高速 LZSS 符号化アーキテクチャ - PAHL-LZSS の構成, " 1996 年電子情報通信学会ソサイエティ大会, D-70, 1996.
- 藤岡豊太, 阿曽弘具, "高速 LZ77 符号化・復号化並列処理アーキテクチャ- PAHL の構成, " 信学技報, vol.95, no.342, CPSY96-69, pp.1-8, Oct. 1996.