平成 14 年度

学士学位論文

並列 BCJR アルゴリズムによる 高速 Turbo-Code 復号方式

A High Speed Turbo-Decoding System based on Parallel BCJR Algorithm

1030315　　宮崎 康徳

指導教員　　岩田 誠 教授

2003 年 2 月 12 日

高知工科大学 情報システム工学科

要 旨

並列 BCJR アルゴリズムによる
高速 Turbo-Code 復号方式

宮崎 康徳

様々な情報システムに依存している近年、情報の信頼性の確保は、現在のみならず、将来を視野に入れても、大容量のデータを扱う情報化社会において、必須である。このような情報の信頼性を確保する技術として、誤り訂正符号は欠くことができない。しかし、伝送される情報は、大容量となるにつれて、誤る可能性は高くなるのが現状である。そこで理論限界に近く、高 BER 特性を持つ誤り訂正符号が必要とされている。現在実用化されている多くの誤り訂正符号は、理論限界に遠く及ばなかった。そこで、1993 年に提案された Turbo-Code は理論限界に最も近いとされ、その潜在的性能の高さは様々な研究で明らかになりつつある。しかし、復号器側では、事後確率を繰り返しにより近似しているため、復号遅延が大きくなり、ソフトウェアによる高速化が困難とされている。

本研究では、このような復号遅延問題を解決するべく、Turbo-Code の並列性について検討し、並列処理能力に優れたデータ駆動型プロセッサによりソフトウェアで高速実現することにより、様々なシステムに柔軟に対応可能で、かつ高速な Turbo-Code 復号処理を実現する。

性能評価の結果、従来の Turbo-Code 復号処理と比較して、本提案手法は処理遅延を削減でき、高速に処理することが可能であることを確認した。

キーワード　Turbo-Code、データ駆動型プロセッサ、DECODER、BCJR アルゴリズム

Abstract

A High Speed Turbo-Decoding System based on Parallel BCJR Algorithm

Yasunori MIYAZAKI

With spreading influence of information systems, guarantee of information reliability is indispensable to information society where a large amount of data is treated. There are error-correcting codes in order to guarantee the information reliability. However, the error rate of the transmiting information becomes higher as data becomes larger. Furthermore, error-correcting codes require the theoretical limit and high BER performance. Many error-correcting codes in practical use are far inferior to theoretical limit.

Turbo-Code suggested in 1993, which is most near to the theoretical limit, has been studied in various research to show the high level of the potential performance. However, in the decoder side, decoding delay becomes large and speeding up by software is difficult for posterior probability approximates by iteration.

In this research, in order to solve such a decoding delay problem, Turbo-Code decoding processing was implemented to be able to respond flexibly in various systems and high-speed processing by data driven processor which realizes high-parallel processing.

As the result of performance evaluation, this proposed method is proved to reduce processing delay and to realize high-speed processing.

key words Turbo-Code, Data-Driven Processor, DECODER, BCJR Algorithm

目次

図目次

表目次

第 1 章

序論

様々な情報システムに深く依存している近年、情報の信頼性の確保は、情報伝送の基盤技術として必須である。ところが伝送される情報が大容量になればなるほど、情報が誤る可能性は高くなる。このため情報の誤りに対する対策は今後の情報化の進展を左右するものの一つといっても過言ではない。

このような情報の誤りに対する対策として、誤り訂正符号が確立されている。誤り訂正符号は復号誤り率、符号化率、計算量などで評価され、その代表的限界には Shannon 限界がある。Shannon 限界とは情報を誤りなく送信できる容量の限界で、現在実用化されている符号の多くは Shannon 限界に遠く及ばなかった。しかし Shannon 限界に近づく符号として、Turbo-Code が 1993 年にフランスの C.Berrou らによって発表された。Turbo-Code は誤り率最小復号を近似的に実現する効率の良い復号アルゴリズムを用いている点に利点があり、Turbo-Code の符号自体の潜在的性能の高さは様々な研究から明らかになりつつある。現在では 2001 年にサービスが開始された W-CDMA にも採用され、注目を集めている。また W-CDMA など無線通信環境に限らず、様々な分野で実用化されている。

しかしそれと同時に Turbo-Code が抱える問題も明らかになってきている。Turbo-Code の特徴である繰り返し復号により、誤り率は低下できるが、復号遅延が大きくなりソフトウェアによる高速化が困難となる問題である。

本稿では、上記の問題を解決するため、Shannon 限界に近い誤り率特性を持つ Turbo-Code を、様々な環境に柔軟に対応可能となるように復号処理を高速化することで、Turbo-Code の更なる優位性を示す。

Turbo-Code 復号処理の高速化へのアプローチとして、復号処理アルゴリズムのストリー

ム並列処理を考える。復号処理の各要素で並列処理可能な部分を抽出し、復号アルゴリズムを、並列処理に優れたデータ駆動型プロセッサにより実現することで、高速化を図る。

データ駆動型プロセッサは、その特徴として、

- 省電力、低発熱
- プログラマブルな高い柔軟性
- 高い並列処理性

などがあげられる。データ駆動型プロセッサはノイマン型プロセッサのように、クロックにより動作するのではなく、処理するデータを扱う部分でのみ電力が消費されるため、低消費電力となる。

またデータ駆動型プロセッサの大きな特徴として、既存のプロセッサとは異なり、データフローグラフでプログラムを記述できる。図 1.1 に既存のプログラムとフローグラフでのプログラムを示す。

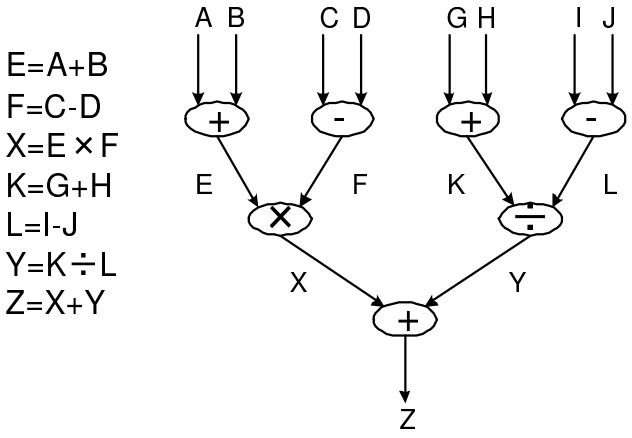

図 1.1　既存のプログラム（左）データフロープログラム（右）

図を見れば明らかなように、同じ処理でも、データフローグラフを用いると、並列処理可能な部分が増加し、その分高速な処理が行える。また、データ駆動型プロセッサは、プログラマブルであるため、様々なシステムへの柔軟な対応も可能となる。このようなデータ駆動型プロセッサは、アプローチとして並列処理性を検討している Turbo-Code 復号処理高速化

の実現とは、非常に相性が良いと考えられる。

以上のアプローチで Turbo-Code の高速化を図り、その優位性を示す。

2 章では、Turbo-Code 復号処理の高速化への問題点を明白にするため、Turbo-Code の符号器、復号器の構成、また処理内容を説明し、問題点を述べる。

3 章では、2 章で定義した問題点の改善策を元に、並列 BCJR アルゴリズムを提案する。並列 BCJR アルゴリズムが理論的に Turbo-Code の復号処理遅延を削減でき、高速処理可能であることを示す。

4 章では、提案した並列 BCJR アルゴリズムを、どのようにデータ駆動型プロセッサによって実現したかを説明し、その問題点、また解決手法を説明する。

5 章では、データ駆動型プロセッサによって実現できた Turbo-Code 復号処理の性能評価を行い、理論値が正しいことを証明する。

6 章では、提案手法、性能評価をもとに考察を行う。

第 2 章

Turbo-Code

2.1　緒言

本章では、Turbo-Code 復号処理の高速実現へ向けて、まず符号器の構成、処理内容を説明する。

次にその処理遅延から Turbo-code のボトルネックとされる復号器の構成、処理内容を説明する。さらに高速実現への問題点を明らかにする。

2.2　ターボ符号器の構成

本節では、ターボ符号器の構成について説明する。図 2.1 にターボ符号器の構成を示す。

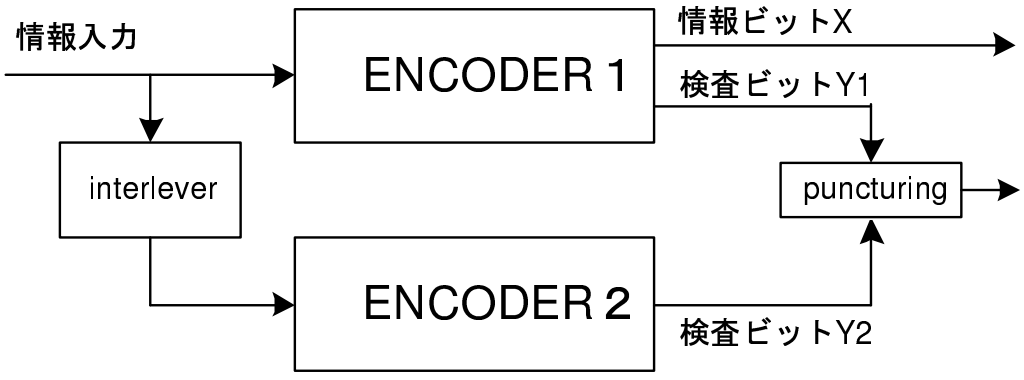

図 2.1　ターボ符号器の構成

2.2　ターボ符号器の構成

ターボ符号は ENCODER（再帰的組織畳み込み符号器）をインタリーバを経由して並列に 2 個連接された構成をとる。但し、インタリーバを介した後の情報ビットは出力されない。インタリーバとは情報系列をランダムにする交錯器で、主に通信路での情報のバースト誤りを解消するために用いられている。

また ENCODER1,2 内部の構成である、再帰的組織畳み込み符号器の構成を図 2.2 に示す。

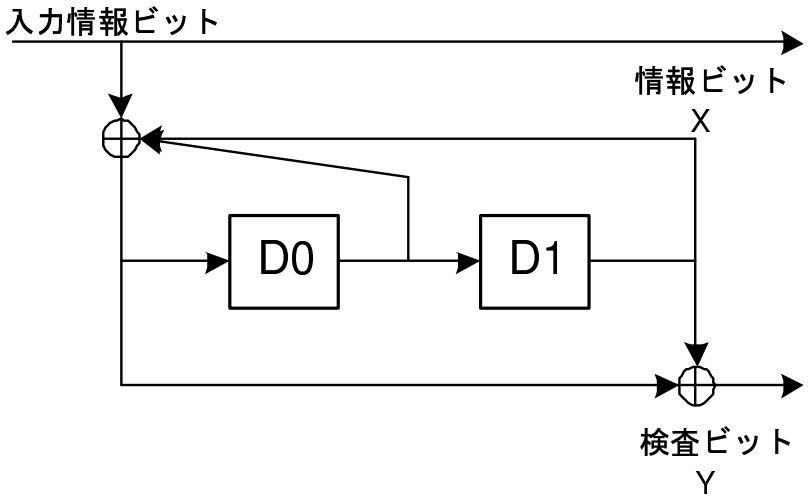

図 2.2　再帰的組織畳み込み符号器の構成

D0、D1 はシフトレジスタであり、ここでは簡略化のためその個数は 2 とした。この畳み込み符号器は、情報ビットと検査ビットが区別されて出力される組織符号である。

入力情報ビットの第 3 番目のビットがシフトレジスタの入力に現れたとき、第 1 番目のビットがシフトレジスタの出力に現れるので、第 3 ビットは過去 2 ビットの影響を受けて符号化される。その影響が再帰的にもなっていることから再帰的組織畳み込み符号と呼ばれている。

また第 3 ビットは過去 2 ビット以前の影響は受けないので、第 3 ビットが符号化にあたって拘束されるのは過去 2 ビットのみである。したがって、シフトレジスタの個数mが符号化においてビットを拘束するので、その個数mを拘束長と呼び、この畳み込み符号器の場合は 2 である。

さらに、ENCODER1,2 によって出力される検査ビット Y1,Y2 をパンクチャリング（間引き）することで、符号器全体の符号化率を様々に変えることができる。例えば検査ビット 1,2 がすべて出力されるのであれば（パンクチャリングなし)、符号化率は $\frac{1}{3}$ であり、そのときの出力系列は

$$\begin{pmatrix} X_1 & X_2 & X_3 & X_4 & \dots \\ Y1_1 & Y1_2 & Y1_3 & Y1_4 & \dots \\ Y2_1 & Y2_2 & Y2_3 & Y2_4 & \dots \end{pmatrix}$$

となる。

また検査ビット 1,2 が交互に出力されるのであれば、符号化率は $\frac{1}{2}$ となり、そのときの出力系列は

$$\begin{pmatrix} X_1 & X_2 & X_3 & X_4 & \dots \\ Y1_1 & Y2_2 & Y1_3 & Y2_4 & \dots \end{pmatrix}$$

となる。

様々な分野で使用されている Turbo-Code は、符号化率、また拘束長などの違いが見られるが、符号化率はこのパンクチャリングで操作し、拘束長はシフトレジスタの個数によって決定される。

このように、符号器の構成要素は ENCODER、インタリーバ、パンクチャリングの３つであり、再帰的組織畳み込み符号器の処理が主であるため、符号処理が単純なものであることは明白であり、符号器に高速実現の問題点となるような部分は特にない。

2.3　ターボ復号器の構成

Turbo-Code の復号器は処理が複雑であり、その処理は反復復号法に基づいている。本節ではターボ復号器の構成について説明する。図 2.3 にターボ復号器の構成図を示す。

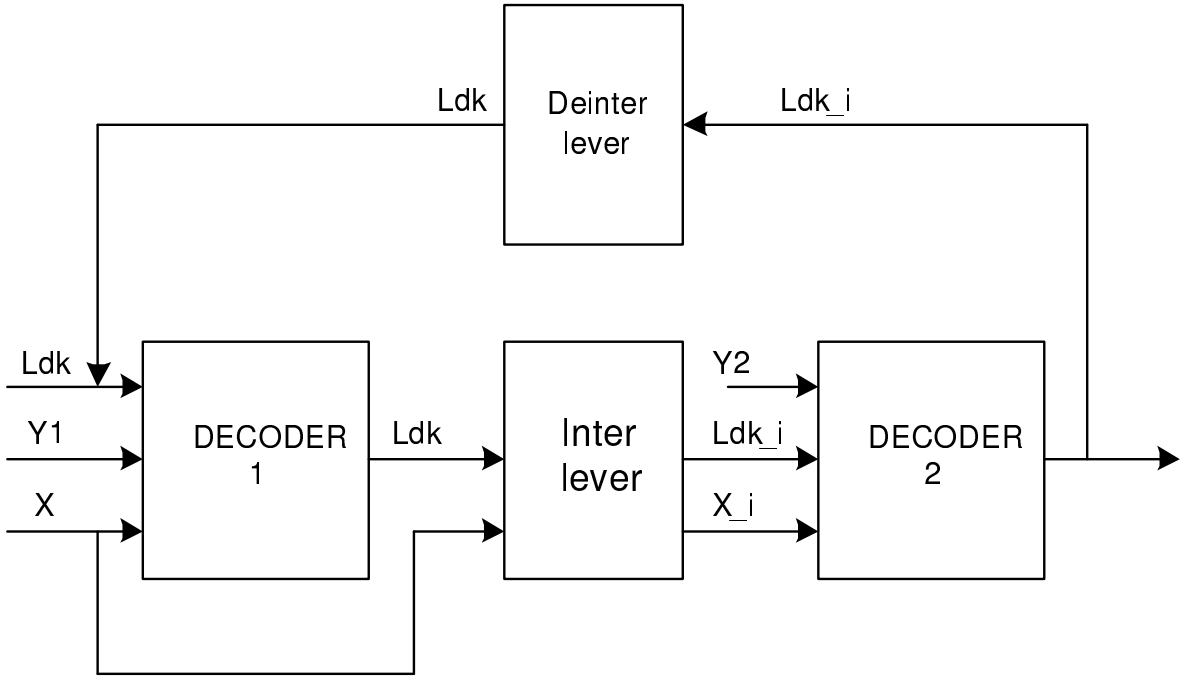

図 2.3　ターボ復号器の構成

復号器は、2 つの DECODER をインタリーバ、デインタリーバを直列に連結した、ループ構成をとる。軟入力、軟出力の復号器で、復号処理として参照されるデータは、情報ビットの信頼度となる信頼度情報ビット系列 Ldk である。入力されてくる情報は前節で説明したうように、符号器を通った情報ビット系列 X、検査ビット系列 Y1、Y2 で、DECODER1 では、情報ビット系列 X と検査ビット系列 Y1 が信頼度情報（1 回目の復号は初期値を与える）を基に復号される。その DECODER1 で得られた新たな信頼度情報は、情報ビット系列 X、と共に符号器と同様のインタリーバで検査ビット Y2 の順に並び替えられ、検査ビット Y2 と並び替えられた信頼度情報 Ldk、情報ビット系列 X が DECODER2 に入力される。DECODER2 で得られた、信頼度情報はデインタリーバによって元の順に戻され、再

度 DECODER1 の入力とされる。このように Turbo-Code の復号は DECODER1 と 2 で信頼度情報をやりとりし、互いに高めあうことで情報を復号する MAP 復号（Maximum A poesteriori Probability decoding）に基づいている。

2.3.1　BCJR アルゴリズム

本項では DECODER の内部処理の BCJR アルゴリズムについて説明する。図 2.4 に DECODER の内部処理のブロック図を示す。

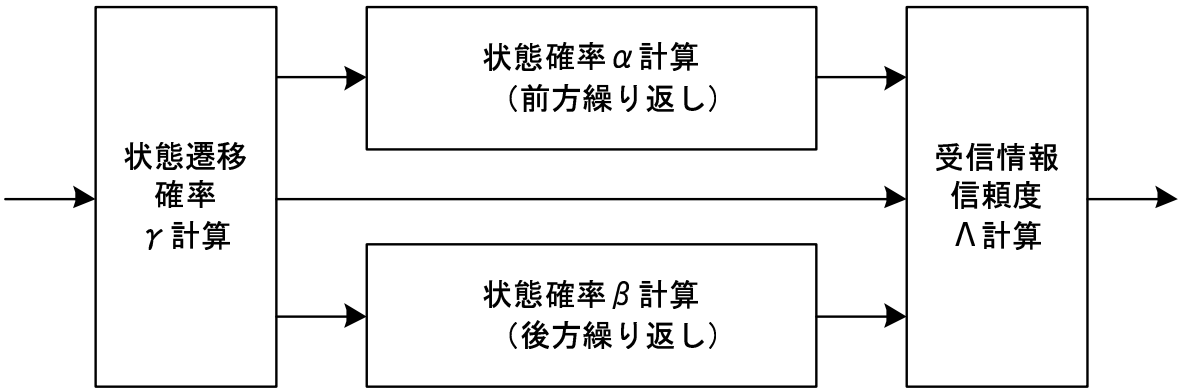

図 2.4　DECODER の内部処理

DECODER の内部処理は BCJR（Bahl-Cocke-Jelinek-Raviv）アルゴリズムに基づいている。BCJR アルゴリズムとは受信情報と信頼度情報から新たな信頼度情報を算出するアルゴリズムで、軟入力、軟出力アルゴリズムである。その処理は主に符号器側での ENCODER 内部処理である再帰的組織畳み込み符号のシフトレジスタの状態確率、状態遷移確率の算出である。まず、受信情報を元に、シフトレジスタの状態遷移確率 γ 計算を行う。その後 γ の値を元に、シフトレジスタの或る状態の確率 α、β 計算を行う。α、β の違いは情報ビット系列の先頭ビットからの状態確立の算出と、終了ビットからの算出の違いである。その後、γ、α、β、3 つの値を使用して、実際に受信情報の信頼度情報となる、Λ 計算を行う。

この BCJR アルゴリズムは、トレリス線図という符号器側でのシフトレジスタの状態変

化の様子を表す状態遷移図で示すと、視覚的に理解し易くなる。図 2.5 に拘束長 2 のトレリス線図を示す。

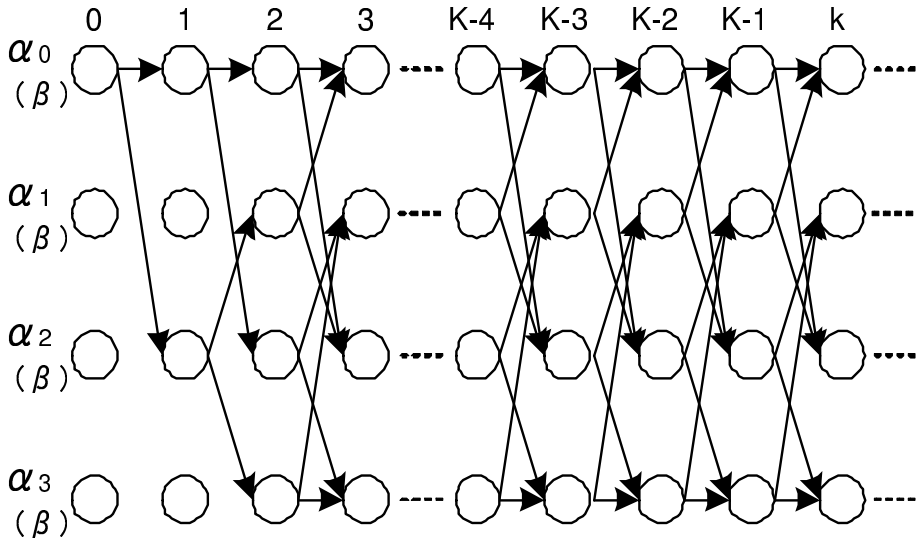

図 2.5　トレリス線図

横軸は時間軸であり、縦軸はシフトレジスタの状態数を表している。拘束長が 2 の場合、シフトレジスタの個数が 2 個であり、2 個のシフトレジスタの状態数は 00、01、10、11、の 4 状態である。4 状態それぞれをトレリス線図では、各々 $\alpha_0(\beta)$、$\alpha_1(\beta)$、$\alpha_2(\beta)$、$\alpha_3(\beta)$ と置いている。例えば 00 というシフトレジスタの状態から 0 が入力されたら状態は 00 のままで、1 が入力されたら状態は 10 に変化する。従って状態 00（α_0）からは矢印が状態 00（α_0）と状態 10（α_2）に伸びている。このトレリス線図を α 計算では前方から行い、β 計算では後方から行う。

このトレリス線図を元に、γ 計算、α 計算、β 計算、Λ 計算処理の具体的な演算を説明する。

2.3.2　状態遷移確率 γ 計算

状態遷移確率 γ 計算の値はトレリス線図では、矢印にあたる。受信信号（情報ビット系列 X と符号出力 Y の組み合わせ 4 種類 00、01、10、11 について）を基に以下の式により状態遷移確率を求める。

例:γ_{00} の場合

$$\gamma_{00} = pdf_0(X) \cdot pdf_0(Y) \cdot (1 - P) \tag{2.1}$$

$pdf_x(r) \cdots$ 受信信号の確率密度。正規分布の確率密度の公式を用いる。平均値は 0 と 1 の 2 種類の場合をとる。

$$pdf_x(r) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left(\frac{(-r - x)}{2\sigma^2}\right) \tag{2.2}$$

$x \cdots$ 信号の平均値（-1、1）

$r \cdots$ 受信値

$P \cdots$ 情報が 1 である事前確率。0 の事前確率は $1 - P$ で表す。

$$P = \frac{\exp(Ldk)}{1 - \exp(Ldk)} \tag{2.3}$$

$Ldk \cdots$ 信頼度情報ビット

式（2.1）を基に受信情報ビット系列 X と検査ビット系列 Y、そして信頼度情報 Ldk を使用して状態遷移確率 γ の値が算出される。

2.3.3　状態確率 α、β 計算

状態確率 α、β 計算の値はトレリス線図では、丸印にあたる。γ 計算で得られた γ の値を基に以下の式で α を算出する。

$$\alpha_0(k) = \alpha_0(k-1) \cdot \gamma_{00}(k-1) + \alpha_1(k-1) \cdot \gamma_{11}(k-1) \tag{2.4}$$

α は時刻 0 から順に時刻 k における状態確率を時刻 k-1 に状態確率と、状態遷移確率 γ により求めていく。$\gamma 0$ の値と $\alpha 0$ を使用して $\alpha 1$ を求め、求めた $\alpha 1$ と $\gamma 1$ の値を使用して $\alpha 2$ を求めるという具合である。

β に関しては時刻 N（N:情報ビット系列）から順に時刻 k における状態確率を k+1 の状態確率と時刻 k の状態遷移確率 γ により求めていく。以下に式を示す。

$$\beta_0(k) = \beta_0(k+1) \cdot \gamma_{00}(k) + \beta_2(k+1) \cdot \gamma_{01}(k) \tag{2.5}$$

以上の算出法を踏まえて α 計算、β 計算の算出過程をブロック図で図 2.6 に示す。

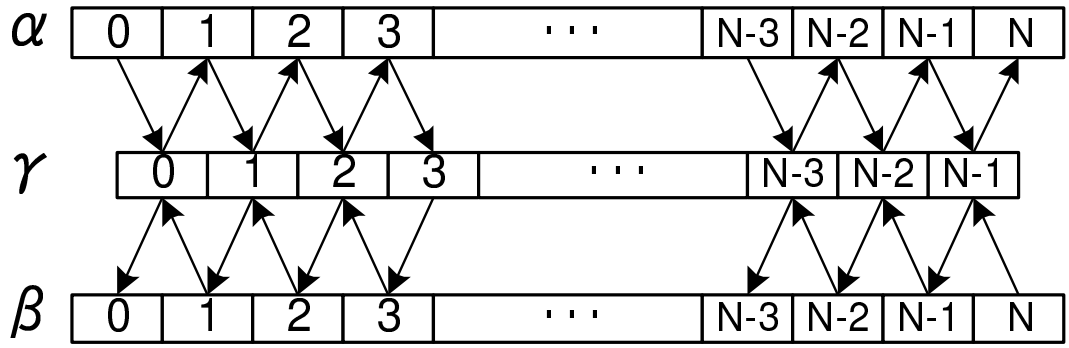

図 2.6　α、β 計算の算出過程

この図で明らかなように、α 計算、β 計算の違いは、状態確率を受信系列の前方から算出するか後方から算出するかの違いである。

2.3.4　受信情報尤度計算 Λ 計算

受信情報尤度計算 Λ 計算はこれまでに求めた γ、α、β の 3 つの値を使用して時刻 k における情報の信頼度を求めていく。以下に Λ の式を示す。

$$\Lambda(k) = \alpha(k) \cdot \gamma(k) \cdot \beta(k+1) \tag{2.6}$$

時刻 k における情報の信頼度なので、時刻 k に関わるすべての状態遷移確率、状態確率を掛け合わせることで求まる。図 2.7 に Λ の算出過程をブロック図でに示す。

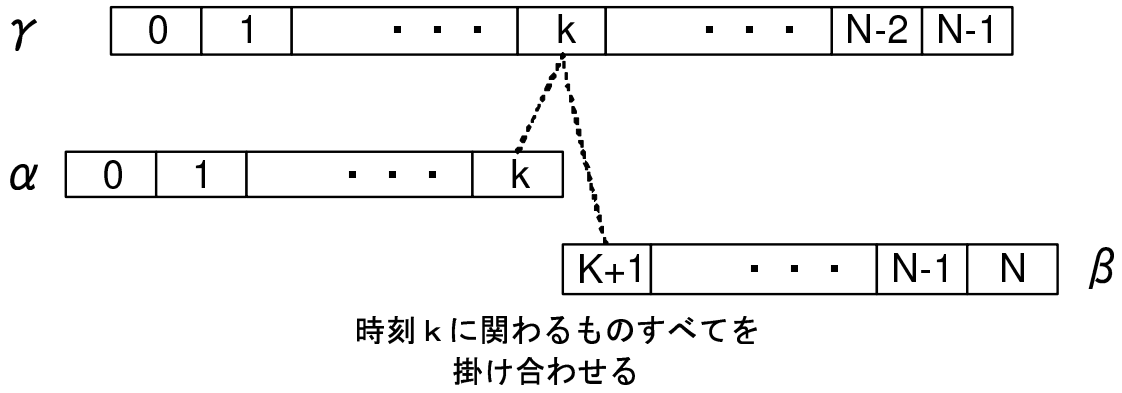

図 2.7　Λ の算出過程

つまり、時刻 0 から算出してきた状態確率と、時刻 N から算出してきた状態確率を時刻 k で挟み込むことにより、信頼度情報を算出している。

このように BCJR アルゴリズムは γ、α、β、Λ と順次求めていくアルゴリズムである。

2.4　インタリーバ・デインタリーバ

前節では、Trubo-Code の複号器の DECODER 内部について説明した。しかし Trubo-Code では DECODER 間にあるインタリーバ、デインタリーバも重要な役割を果たしており、本節ではインタリーバ、デインタリーバの具体的処理内容について例をとって説明する。

前節で少し述べたが、インタリーバは主に通信路でのバースト誤りを解消するためのものである。バースト的な誤りも情報ビットの順序を並び替えることにより、ランダム的な誤りになる。Trubo-Code はランダム誤りには強いが、バースト誤りに弱いという特質を考慮し、使用されている。

したがって復号器側でも符号器側でインタリーブされた検査ビットを参照する復号では、受信情報ビット、信頼度情報ビットもインタリーブする必要があり、そのインタリーブも符号器側と同様のインタリーバを使用しなければならない。また Trubo-Code の復号処理は繰り返しであるため、インタリーブされた情報を再び元に戻す、デインタリーバの必要性も明白である。

インタリーブ手法には様々なものがあり、その選び方によって BER 特性も変動するため、インタリーバの構成法について様々な研究がなされている。最も一般的な例を上げると、m × n ビットの情報系列を考えた場合、縦方向に書き込み、横方向に読み込む *BlockInterleaver* などがある。図 2.8 に *BlockInterleaver* の概念図を示す。

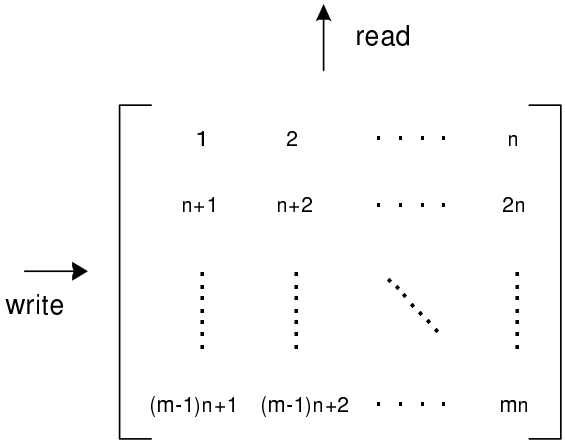

図 2.8　*BlockInterleaver*

2.5　復号処理における高速実現への問題点

前節で説明した BCJR アルゴリズムのボトルネックとなる特徴は、要素毎の逐次処理である。特に α 計算、β 計算においては γ 全ての値が必要であるため、大きな処理遅延が発生する。事前に並列処理可能な部分がある環境では必ずしも最適ではない。

したがって並列処理可能部分について検討する必要がある。また各要素の処理遅延の削減の検討も行う必要がある。

2.6　結言

本章では、Turbo-Code 復号処理の高速実現への問題点を明らかにするため、まず Turbo-Code の符号器について説明した。次に復号処理について BCJR アルゴリズムを説明し、その処理が逐次的であるという問題を定義し、Turbo-Code の高速実現にはこの問題の改善の必要性があることを示した。

第 3 章

並列 BCJR アルゴリズム

3.1 緒言

本章では、前章で考察した Turbo-Code 復号処理の高速実現の問題点をうけ、まずその改善策を考察する。次にその改善策で具体的な手法である、並列 BCJR アルゴリズムを提案する。並列 BCJR アルゴリズムの詳細を説明し、理論的に処理遅延が削減できることを示す。

3.2 問題点の改善策

前章で示したように高速実現の問題点として、逐次処理が挙げられる。復号処理の高速実現には、

- 部分演算処理可能な部分の並列処理を増加させる
- 各要素毎のクリティカルパスを可能な限り短縮させる

という、2 点が必要である。そこでこの 2 点について具体的な改善策を考察すると、

- γ 計算の終了を待たなくても部分的にでも α 計算、または β の処理を開始する
- α 計算、β 計算の並列処理
- 各要素の内部を可能な限り並列処理する

等が挙げられる。

3.3　並列 BCJR アルゴリズム

本節では、前節で上げた改善策から処理遅延が削減可能な並列 BCJR アルゴリズムによる高速実現手法を提案する。

並列 BCJR アルゴリズムは α 計算式の展開を行うことで実現できる。式（2,4）の α 計算式だが、この式は漸化式となっている。式（2,4）では $\alpha_0(k)$ しか示していないが、他の状態の $\alpha_1(k)$、$\alpha_2(k)$、$\alpha_3(k)$ を求める式もあわせてもう一度示すと以下のようになる。

$$\alpha_0(k) = \alpha_0(k-1)\cdot\gamma_{00}(k-1) + \alpha_1(k-1)\cdot\gamma_{11}(k-1) \tag{3.1}$$
$$\alpha_1(k) = \alpha_2(k-1)\cdot\gamma_{10}(k-1) + \alpha_3(k-1)\cdot\gamma_{01}(k-1) \tag{3.2}$$
$$\alpha_2(k) = \alpha_0(k-1)\cdot\gamma_{11}(k-1) + \alpha_1(k-1)\cdot\gamma_{00}(k-1) \tag{3.3}$$
$$\alpha_3(k) = \alpha_2(k-1)\cdot\gamma_{01}(k-1) + \alpha_3(k-1)\cdot\gamma_{10}(k-1) \tag{3.4}$$

これらの式を使用して式を展開すると、以下のような $\alpha_0(k)$ が得られる。

$$\begin{aligned}\alpha_0(k) = \alpha_0(k-2)&\underline{\gamma_0 0(k-2)\gamma_0 0(k-1)}\\ +\alpha_1(k-2)&\underline{\gamma_1 1(k-2)\gamma_0 0(k-1)}\\ +\alpha_2(k-2)&\underline{\gamma_1 0(k-2)\gamma_1 1(k-1)}\\ +\alpha_3(k-2)&\underline{\gamma_0 1(k-2)\gamma_1 1(k-1)}\end{aligned} \tag{3.5}$$

この式の利用により、$\alpha(k-2)$ が求まっていない時点でも、$\gamma(k-1)$ と $\gamma(k-2)$ の値が確定し次第、下線部の部分積は開始可能で、その後 $\alpha(k-2)$ が確定後は内積演算 1 回で $\alpha(k)$ を計算できる。

さらに α 計算と β 計算の処理が同様であることから、この手法は β 計算においても利用可能である。これは前章のトレリス線図で説明するとトレリス線図のグループ化ということになる。トレリス線図のグループ化の概念図を図 3.1 に示す。

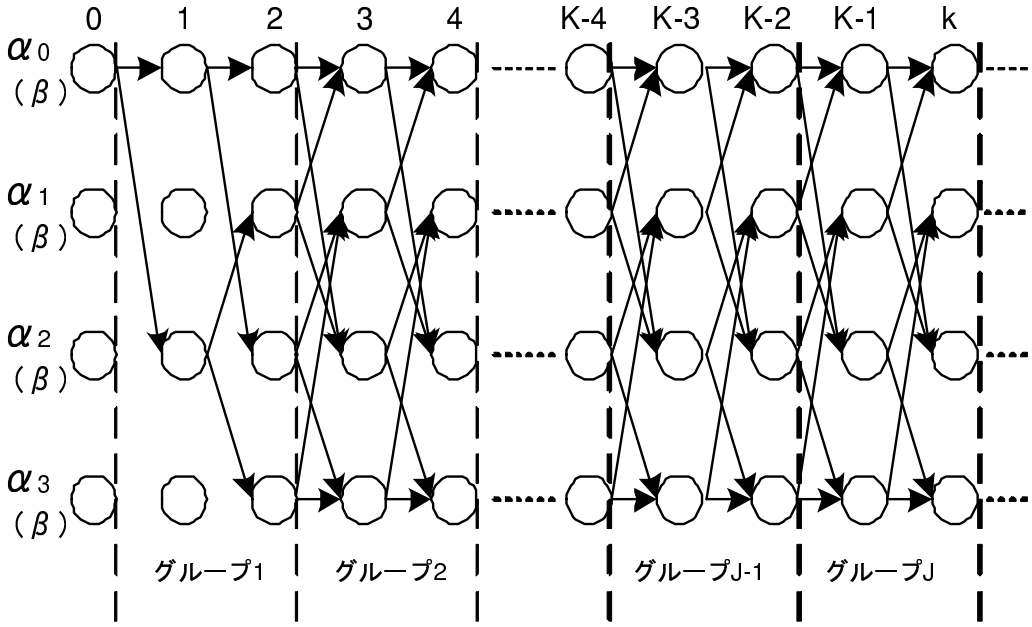

図 3.1　トレリス線図のグループ化

矢印である γ の部分積を先に計算しておくことで、一回の演算で 2 つの丸印 α を求めることができる。

この提案手法の利点は、γ の値を求めている最中に並列に部分積を求めておくことで、α 計算、β 計算の処理遅延を削減しようとする点にある。α 計算、β 計算では掛算処理が一回でよく、さらにその一回の演算で 2 つの α または β が求まるために、$\alpha(1)$ から $\alpha(N)$ を求める総時間は従来方式の NT から $\frac{NT}{G}$ に削減可能である。ここで N は情報ビット系列、T は 1 つの α を求める時間、G はグループ内サイズである。

3.4　結言

本章では、前章で定義した Turbo-Code 復号処理の高速化の問題点を改善し、処理遅延を削減可能な、並列 BCJR アルゴリズムを提案した。この並列 BCJR アルゴリズムにより、トレリス線図のグループ化ができ、γ 計算中に部分並列処理が可能となった。その結果処理遅延は従来方式と比較して $\frac{1}{G}$ に削減可能であることを示した。

第 4 章

データ駆動型プロセッサによる実現

4.1 緒言

本章では前章で定義した、並列 BCJR アルゴリズムの最適な構成について考え、データ駆動型プロセッサによる実現法を示す。

まず Turbo-Code 復号処理のそのままデータ駆動型プロセッサでプログラミングする。次にそのプログラムの問題点を考察し、最適な構成となるよう改良する。

4.2 データ駆動型プロセッサでの実現

ここでは、提案手法の Turbo-Code 復号処理をデータ駆動型プロセッサで実現するにあたり、DECODER の構成と、インタリーバの構成を説明する。

4.2.1 DECODER の構成

Turbo-Code 復号処理の DECODER をデータ駆動型プロセッサでプログラミングすると、図 4.1 のフローグラフになる。

各要素から出力される 4 つの枝は、状態数である。拘束長は 2 であるため、その状態数は 00、01、10、11 の 4 つになる。各々の状態について γ、α、β の値を算出する。

まず入力情報となる、情報ビット系列 X、検査ビット系列 Y1、Y2 をメモリに格納する。次に γ 計算部で、情報ビット系列 X と検査ビット系列 Y1 を読み出し、読み出した値と信頼度情報 Ldk をもとに γ 計算を行う。信頼度情報 Ldk は初回の復号は初期値 0.5 を与える。

初期値 0.5 とは、1 か 0 か 50 %の確率であるということである。

次に γ 計算後の値をメモリに格納する。ここで、提案手法プログラムに関しては、γ の値と、γ の部分積の値の両方を格納する。この γ 値格納部では一旦全ての値が格納されるまで待つ。その終了待ちは、入力情報の最終ビットに終了判定ビットを加えることで行う。終了判定ビットが入力されたら、次の処理を開始する。終了判定ビットは、次の α 計算、β 計算の初期値に変化し、格納した γ の値を使用しながら α、β を算出する。

最後に、算出された γ、α、β の値を基に、Λ 計算を行う。出力された Ldk はインタリーバによって順番を入れ替え、その系列の最後にもう一度終了判定ビットを付加し、次の DECODER2 の入力とされる。

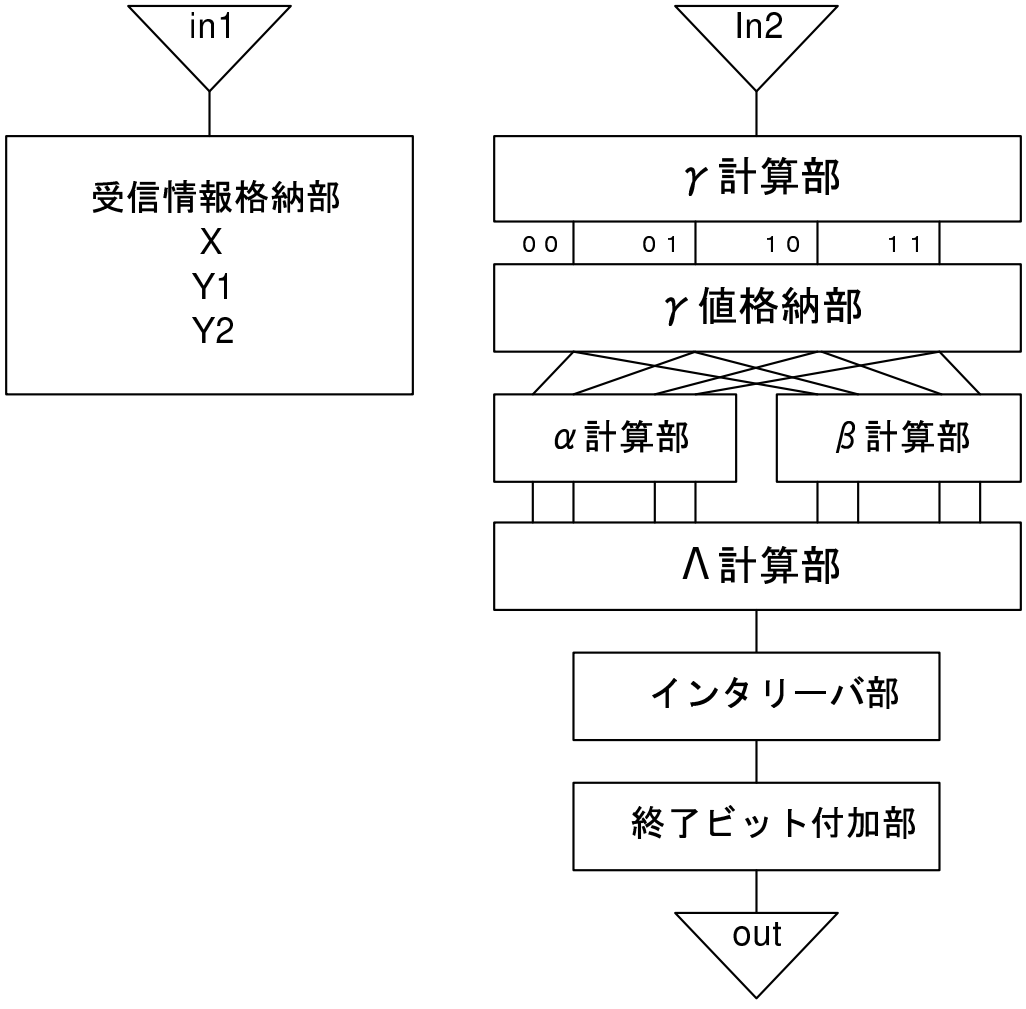

図 4.1　要素復号器のデータフローグラフ

4.2.2　インタリーバの構成

Turbo-Code 復号処理において、もう一つ重要となる要素はインタリーバである。このインタリーバをデータ駆動型プロセッサで構成する際、2.4 で示した、*BlockInterleaver* を用いた。またインタリーバをデータ駆動型プロセッサで構成すると、ランダムにするべきデータは世代番号で行える。例えば、4×4*BlockInterleaver* を考えると、世代番号の変化は

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 0 & 4 & 8 & 12 & 1 & 5 & 9 & 13 & 2 & 6 & 10 & 14 & 3 & 7 & 11 & 15 \end{pmatrix}$$

となる。これを 2 進数表記すると、

$$\begin{pmatrix} 0000 & 0001 & 0010 & 0011 & 0100 & 0101 & 0110 & 0111 & & & & \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & & & & \\ 0000 & 0100 & 1000 & 1100 & 0001 & 0101 & 1001 & 1101 & & & & \\ & & & & 1000 & 1001 & 1010 & 1011 & 1100 & 1101 & 1110 & 1111 \\ & & & & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ & & & & 0010 & 0110 & 1010 & 1101 & 0011 & 0111 & 1011 & 1111 \end{pmatrix}$$

となる。これは、世代番号を 4 ビットとみなし、下位 2 ビットと、上位 2 ビットを交換していることになる。そこで、これに着目し、データ駆動型プロセッサで *BlockInterleaver* を構成すると、従来インタリーバのために必要とされたメモリを全く使用せずに構成できる。*BlockInterleaver* をデータフローグラフで表現すると図 4.2 のようになる。

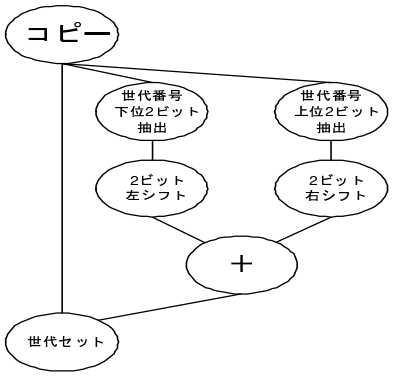

図 4.2　インタリーバのデータフローグラフ

この手法を用いると、このような非常に簡単なインタリーバの構成がとれ、またメモリも

必要でないため、非常に効率が良いといえる。

4.3　問題点

ここでは、提案手法における問題点について考察し、その解決策を検討する。

提案手法の考えらえる問題点として、γ の部分積格納部での部分積を計算する際の処理負荷や、$\alpha\beta$ 計算の並列処理による処理負荷の増大が考えられる。データ駆動型プロセッサは、並列処理が可能で、高速にデータ処理が行えるが、並列処理の増大は高速データ処理が行えると同時に、処理負荷の増大にも繋がる。$\alpha\beta$ 計算部は特に、前方から演算するか、後方から演算するかの違い以外はほぼ同様の処理を行っているのにも関わらず、逐次処理によって 2 つとも各処理負荷は大きい。よって処理負荷を軽減する手法の提案の必要性がある。処理負荷の軽減案としては、演算命令数の削減が考えられる。そこで、命令数の削減によって処理負荷を軽減する手法を提案する。

4.4　問題点の解決策

4.4.1　2 世代化

処理負荷の軽減において命令数の削減を可能にするため、データ駆動型プロセッサのハードウェア機構の 2 世代化と呼ばれる、1 つのパケットに 2 つのデータを格納し、それらを同時に処理する方式を使用する。簡単なデータ駆動型プロセッサの 2 世代化入力パケット構成を図 4.3 に示す。

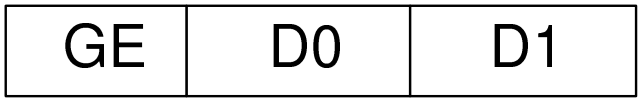

図 4.3　入力データパケット構成

GE は、世代番号と呼ばれる領域で、プログラムにストリーム状のデータが流れ込んできた場合に、データそれぞれを識別するためにつけられる。世代番号がマッチングするパケッ

ト同士で命令の演算を行う。

D0、D1 はデータ本体の領域で、実際に処理するデータを格納する。1 世代パケットでは D0 の領域のみにデータを格納するのに対し、2 世代化パケットは D0 と D1 にデータを格納し１パケットととして処理される。

このようなパケット構成をとるデータ駆動型プロセッサでの、1 世代パケット処理と 2 世代パケット処理の処理負荷の違いを、図 4.4 で加算命令の実行を例にとり説明する。

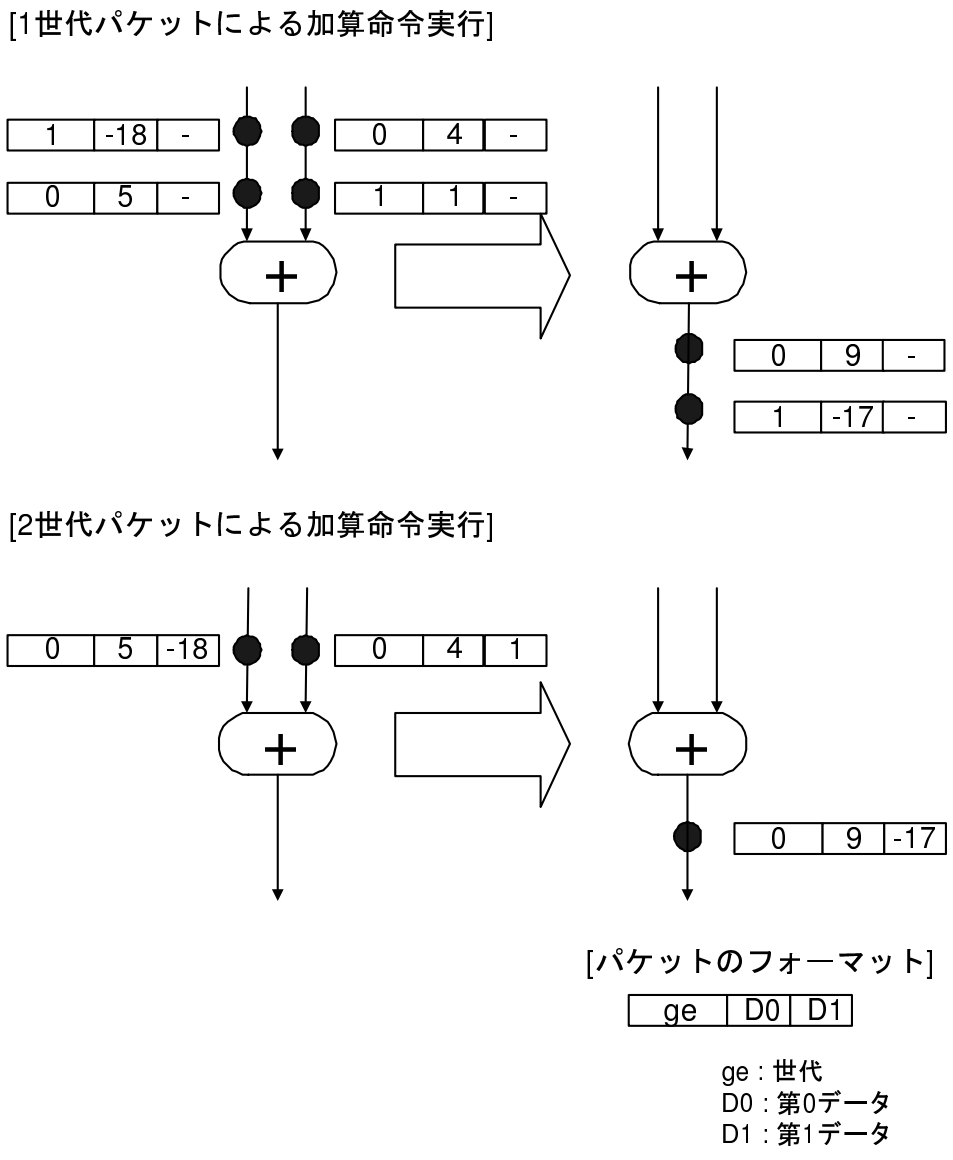

図 4.4　1 世代パケット処理と 2 世代パケット処理の比較

図から明らかなように、2 世代化を行うと、同数のパケットを処理する場合、パケット数は 1 世代と比べて半分となる。これは、入力データパケットが半分となることを意味し、処理負荷の削減も行えると同時に、処理遅延の削減にも繋がる。Turbo-Code はブロック符号であり、ブロックサイズは 5000 ビットのような大きな値となる場合もあるため、データ数が半分となることは、大幅な処理負荷、処理遅延の軽減に繋がる。

この 2 世代化を Turbo-Code 復号処理に用いるだけでも、更なる処理遅延の削減、処理負荷が軽減が見込める。また、$\alpha\beta$ 計算の処理が同様という特徴を利用して、2 世代化パケットの D0、D1 の領域に、D0 には α の値を、D1 には β 計算の値を格納することで、$\alpha\beta$ 計算の同時計算が可能となり、β 計算部の命令数が全て削減できる。このことにより、さらに処理負荷は軽減できるが、そのための命令の追加が余儀なくされる。そこで、2 世代化向けの簡単な新命令を追加することにより、その問題を解決した。

4.4.2　2 世代化処理向け新命令

2 世代化処理向け新命令は、

- sep 命令
- comb 命令
- vn-rg 命令

の 3 つである。

sep 命令

2 世代化を行うことにより、処理遅延、処理負荷の軽減は実現できる。しかし、データを 2 世代化したままではインタリーバの処理が行えない。インタリーバは完全に入力順序をランダムにする機構であるが、2 データが 1 パケットになっていては、ランダムになるのはパケットであるため、パケット内部のデータまでは完全にランダムにはならない。そこで、2

世代化パケットを一旦 1 世代パケットに戻して処理する必要がある。この sep 命令はそのための命令で、2 世代化パケットを 1 世代にする命令である。動作を図 4.5 に示す。

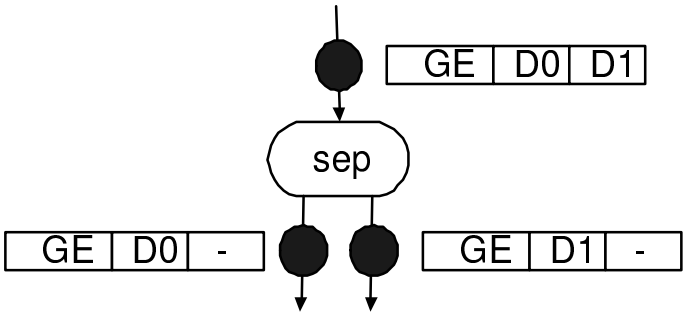

図 4.5　sep 命令

1 入力 2 出力命令で、入力データの D0 を D0 にセットし、世代番号はそのままで左出力とし、入力データの D1 を D0 にセットし、世代番号はそのままで、右出力とする。左右出力パケットの両方とも D1 パケットはない。

comb 命令

この命令は、sep 命令と対をなす命令で、1 世代パケットを 2 世代化パケットにする命令である。インタリーバ等で 1 世代となったデータは次の要素の入力とするために、再び 2 世代化を行う必要がある。動作を図 4.6 に示す。

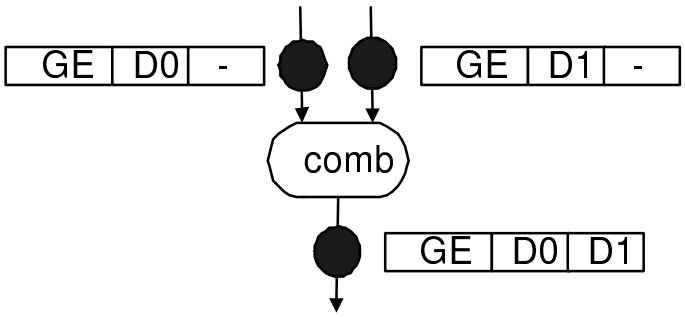

図 4.6　comb 命令

2 入力 1 出力命令で、左入力と右入力の 1 世代パケットを 2 世代化し、左入力データを

D0 に、右入力データを D1 にセットし出力する。この comb 命令と sep 命令を使用することで、容易に 1 世代化、2 世代化の変更が可能となる。

vn-rg 命令

Turbo-Code 復号処理にメモリ操作が非常に多く含まれるため、効率的なメモリアクセスを可能とする命令が必要となる。そのアクセス方式はいくつかあり、世代番号をアドレスとし、アクセスする手法や、右データをアドレスとしアクセスする手法などがある。Turbo-Code としては前者が有効であるが、Turbo-Code はデータがブロックとして入力されてくるため、データ駆動型プロセッサでは世代番号に、データ識別用と、ブロック識別用の領域を確保し処理している。従来のデータ駆動型プロセッサではデータ識別用の領域しかメモリアクセスの際に参照しなかったため、この vn-rg 命令は、データ識別領域と、ブロック識別領域を参照しアクセスできるものとした。さらには、右データも参照することでいかなるアドレスへのメモリアクセスが可能となる。動作を図 4.7 に示す。

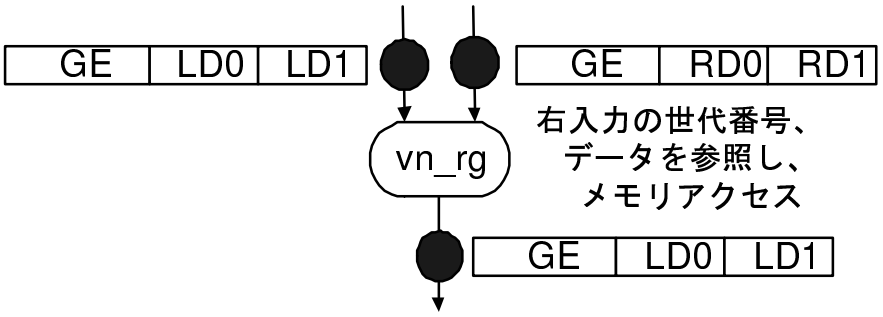

図 4.7　vn-rg 命令

2 入力 1 出力命令で、左データを右データの世代番号、データを参照し、メモリアクセスを行う。メモリアクセスには参照、書き込み、また参照&演算などがあり、メモリアクセス命令全てに関して、この新命令が適用できるものとするため、命令に-（ハイフン）を入れている。例えば、メモリ書き込み命令にこの命令を適用すると、vnreprg 命令となり処理される。

以上の新命令を含んだ 2 世代化提案手法と、1 世代提案手法の処理負荷を比較すると表 4.1 のような結果が得られた。

表 4.1　1 世代化と 2 世代化の処理負荷の比較

	処理負荷 (正規化)
1 世代	1.00
2 世代	0.84

これにより、2 世代化は新命令を含む処理のオーバーヘッドがあるものの、約 26 %の削減となり、2 世代化による処理遅延の削減も見込めるため、非常に有効であることが分かった。

4.5　結言

本章では前章で定義した、並列 BCJR アルゴリズムの最適な構成について考え、データ駆動型プロセッサによる実現法を示した。

まず Turbo-Code 復号処理の DECODER、インタリーバをデータ駆動型プロセッサで構成し、プログラムの問題点を考察したところ、並列処理による処理負荷の増大が考えられた。そこで処理負荷の軽減となるべく、データ駆動型プロセッサの 2 世代化を使用し、処理負荷の軽減を行った。結果、2 世代化を適用することにより、処理負荷を軽減できることを示した。

第 5 章

シミュレーション性能評価

5.1 緒言

本章では、前章までに示した、並列 BCJR アルゴリズムの有効性を示すために、データ駆動型プロセッサによる、性能評価を行う。

まず比較対象となるプログラムを示す。次に比較対象となるプログラムにおいて処理遅延を、従来方式との比較、1 世代化と 2 世代化の比較による見積もり結果を示す。見積もりというのは、実際に新命令が作成されてないため、正しい評価ができないためである。

次に新命令を他の命令に置き換えた、処理遅延のみを評価できるプログラムによる性能評価をデータ駆動型プロセッサシミュレータ上で行い、見積もりの妥当性を証明する。

5.2 比較対象

ここでは、比較対象となるプログラムの説明を行う。比較対象としては、

- 提案手法
- γ の部分積の事前演算は行わず、グループ化のみを適用する手法
- 従来方式

の 3 方式とした。

提案手法は、前章までに示した、並列 BCJR アルゴリズムの実装である。

グループ化のみを適用する手法は、γ の部分積の事前演算は行わず、単純にグループ化のみを適用する手法で、α、β 計算における逐次処理のループ構成の内部を 2 段階にするこで

行っている。この手法は、並列処理の有効性を示すために比較対象とした。提案手法と比較して内部クリティカルパスが若干長い。

従来方式は、従来の BCJR アルゴリズムを実装したものである。

3 方式の大きな違いは、γ 格納部と、α、β 計算部である。γ 格納部は提案手法は、部分積を格納する処理があり、他 2 つの手法は、γ 値の格納のみである。α、β 計算部では、ループ構成の違いがあり、提案手法は、2 状態を 1 回のループで求め、さらに、2 状態それぞれを並列処理している。グループ化のみを適用する手法では、2 状態を 1 回のループで求めているが、並列処理をしていない。従来方式は 1 状態を 1 回のループで求めている。3 方式の α、β 計算部におけるループ構成の概念図を図 5.1 に示す。

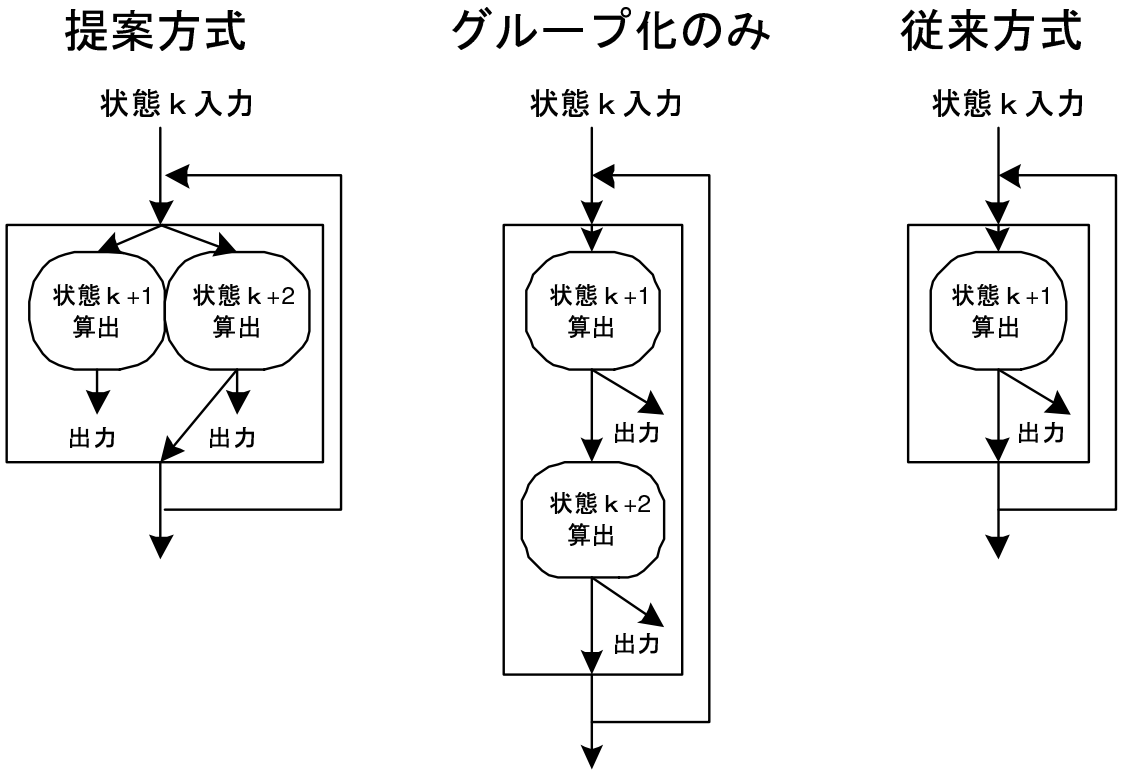

図 5.1　α、β 計算部におけるループ構成

5.3　性能評価理論値

ここでは比較対象とした 3 方式における、1 世代化、2 世代化との処理遅延の性能見積もり結果を示す。評価は DECODER 部、インタリーバ部、終了付加部を対象に行っている。つまり、一回の復号における半分の処理である。残りの半分は同様の処理であるため、スケーリングにより求めることが可能となるからである。図 5.2 に処理遅延の性能評価グラフを示す。

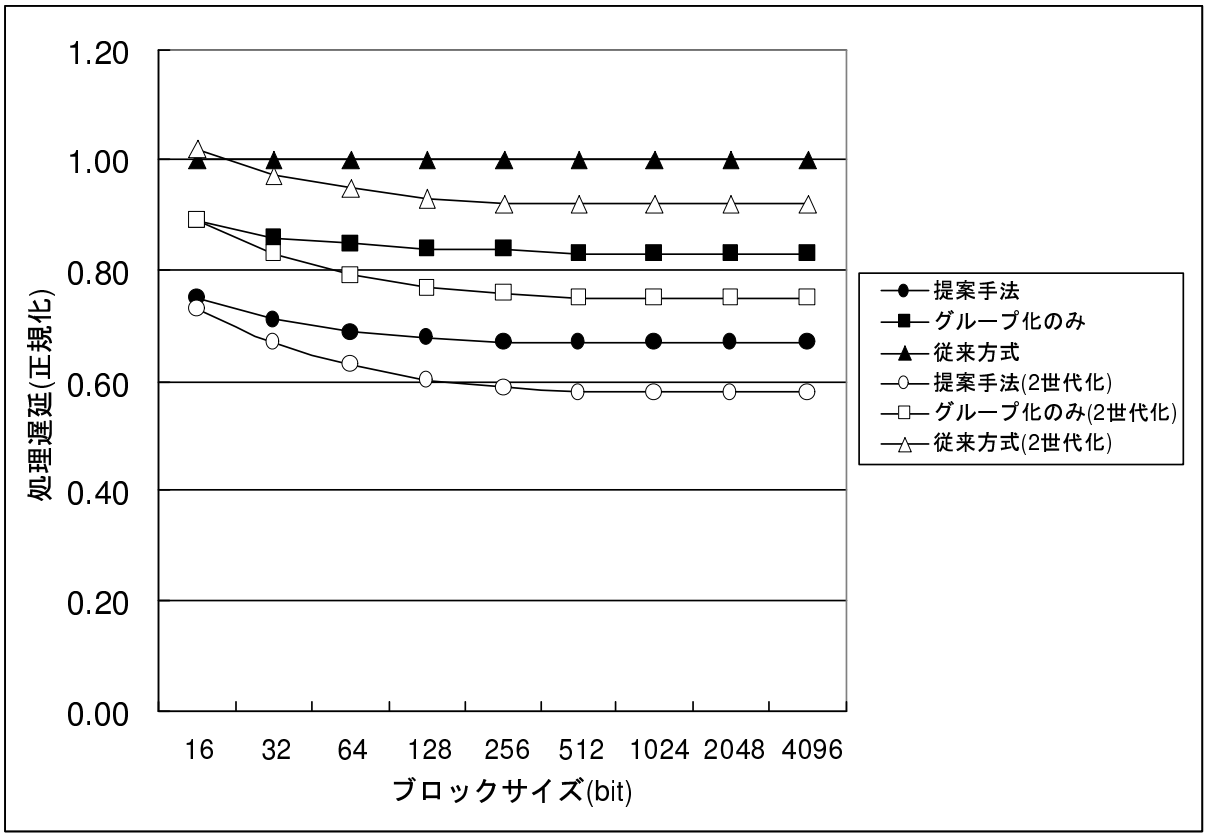

図 5.2　性能見積もり結果による比較

縦軸が処理遅延、横軸がブロックサイズ (bit) で示している。処理遅延は全て正規化している。1 世代の従来方式を 1.00 とし、他の方式との比較を行っている。このグラフから、ブロックサイズが大きくなる程処理遅延の削減は大きくなることが分かる。また、それと同時に処理遅延の削減割合はある値に収束していることも分かる。収束値を表 5.1 に示す。

表 5.1　処理遅延の収束値

	従来手法	グループ化のみ	提案手法
1 世代化	1.00	0.83	0.67
2 世代化	0.92	0.75	0.58

なぜ収束するかについては、処理遅延の見積もり法による。今回処理遅延の見積もりはプログラムのクリティカルパスを比較することで行った。Turbo-Code のクリティカルパスはブロックサイズに完全に依存している。以表 5.2 に各手法のクリティカルパスを示す。

表 5.2　各方式のクリティカルパス

	従来手法	グループ化のみ	提案手法
1 世代化	27+6B	28+5B	28+4B
2 世代化	37+5.5B	38+4.5B	34+3.5B

B:ブロックサイズ

上記を見ても明らかなように、どの手法においても B に依存している。B を∞にすると、その比は B の係数となり、収束する。

これで従来手法と比較して、提案手法は約 33 %、さらに 2 世代化を用いると、約 42 %の処理遅延の削減が可能であることが分かった。

5.4　見積もり結果の証明

これまでの見積もりは、新命令のため、プログラムが作成できないために行っていた。よってその見積もりを証明する必要がある。ここでは、新命令を他の命令に置き換えることで、正しい結果は得られないが、処理遅延のみは評価できるプログラムを作成し、その評価を行う。その結果と見積もり結果を比較することで、見積もりが正しいことを証明する。

まず新命令を置き換えた命令についてだが、sep 命令は、1 入力 2 出力で、入力データと出力データはほぼ同等なものと考えられるので、入力データを複製する copy 命令に置き換

えた。comb 命令は、2 入力 1 出力で、データを合わせる操作を行っているため、加算命令と置き換えた。vn-rg 命令は、メモリアクセス命令であるため、メモリアドレスを世代番号を参照しアクセスする、vn-g 命令と置き換えた。

以上の置き換えを行い、プログラムを作成し、ブロックサイズを 16 ビットとし、データ駆動型プロセッサのシミュレータで処理遅延を算出したところ、表 5.3 のような結果が得られた。

表 5.3　処理遅延評価用のプログラムでの性能評価

	従来方式	グループ化のみ	提案手法
1 世代化	$\frac{37.5\mu s}{(1.00)}$	$\frac{34.1\mu s}{(0.90)}$	$\frac{31.6\mu s}{(0.84)}$
2 世代化	$\frac{38.0\mu s}{(1.01)}$	$\frac{34.0\mu s}{(0.91)}$	$\frac{28.5\mu s}{(0.76)}$

ブロックサイズ 16 ビットの性能見積もりは表 5.4 のようになる。

表 5.4　ブロックサイズ 16 ビットの性能見積もり結果

	従来手法	グループ化のみ	提案手法
1 世代化	1.00	0.89	0.75
2 世代化	1.02	0.89	0.73

これらを比較すると、多少の誤差は見られるものの、ほぼ同等の結果が得られていることが分かる。よって性能見積もり結果は正しいことが証明された。

5.5　結言

本章では、並列 BCJR アルゴリズムの優位性を示すために、データ駆動型プロセッサによる、性能評価を行った。まず性能見積もり結果を出し、その見積もりが正しいかどうかを、ダミープログラムによって証明した。結果、提案手法を用いると、最大で約 42 %の処理遅延の削減が可能であることを示した。

第 6 章

結論

これから更なる進化をとげるであろう情報システムでは、情報の信頼性の確保というものは、欠かせない技術の一つである。その技術が誤り訂正符号であり、現在最も理論限界に近いとされる Turbo-Code は、様々なシステムに活用されるなどその優位性を示している。しかし、それと同時に復号処理遅延の増大という問題があり、様々な研究が行われている。本研究では、Turbo-Code の復号処理遅延問題を解決するべく、Turbo-Code の並列性を検討し、提案手法により処理遅延の削減を行った。

また、並列処理能力に優れたデータ駆動型プロセッサでソフトウェアによって実装することにより、様々なシステムに柔軟に対応可能となることを示した。結果、提案手法である並列 BCJR アルゴリズムを用いると、最大 42 %の処理遅延の削減が可能であり、その優位性を示した。

今後の検討課題は以下の通りである。

- メモリ使用量の削減

これは、提案手法が γ の部分積をあらかじめ計算しておくことによる、部分積の格納領域が余分に必要であるためである。

Turbo-Code をデータ駆動型プロセッサで実装するために必要なメモリ領域として、入力情報系列、γ 値、がある。入力情報は受信値で、受信値を基に復号し、復号回数分、受信値を使用するため必要であり、また γ 値は DECODER の最終要素である Λ 計算において γ の値も参照するために必要である。そこで提案手法と従来手法のメモリ使用量を比べると、表 6.1 のような結果が得られた。

表 6.1　提案手法と従来方式のメモリ使用量の比較

	入力情報系列格納	γ 値格納	合計
提案手法	4B	4B	8B
従来方式	4B	4B+8B	16B

B:ブロックサイズ

入力情報系列格納の際は、情報ビット系列 X、検査ビット系列 Y1、Y2、またインタリーブされた情報ビット系列 X_2 の 4 系列分をブロックサイズ分必要であるため、4 × B となる。また γ 値格納については提案手法、従来手法ともに、4 状態分系列を格納する必要があるため、4 × B、また提案手法については、部分積の格納のため、8B 余分に必要となる。これにより提案手法は従来方式と比較して、2 倍程度のメモリ領域が必要となることが分かる。よってメモリの削減手法が必要となってくる。これについては、BCJR アルゴリズムはメモリを必ず使用するため、他のアルゴリズムを検討する必要があると考えている。

- グループ内サイズ 3、4... の検証

今回提案した並列 BCJR アルゴリズムは、トレリス線図をグループ化することにより処理遅延を削減するもので、そのグループ内サイズは 2 とした。しかし、グループ内サイズを 3、4... と増やせば、それだけさらに処理遅延は削減できる。だが、その際の部分積の格納もさらに必要となってくるため、これについては、上記のメモリ使用量の削減といっしょに考察する必要がある。メモリ使用量とのトレードオフを考えながら、グループ内サイズを増やしての検証が必要となる。

- 実機による検証

今回、評価を行ったのはデータ駆動型プロセッサのシミュレータである。よって実機による実測値での評価を検討する必要がある。今回使用したシミュレータは、アーキテクチャレベルでのシミュレータであるため、実機での検証結果も誤差は少ないものと考えている。

- 分割 BCJR アルゴリズムとの併用による検証

自ら提案した、並列 BCJR アルゴリズムが、トレリス線図をグループ化し、削減する手法に対し、分割 BCJR アルゴリズムは、トレリス線図全体を幾数個に分割し、並列に処理することで処理遅延を削減する手法である。分割 BCJR アルゴリズムは、トレリス線図を分割するため、分割した中間ブロックの初期値を前回の復号結果としている。そのためある程度の性能劣化が見られる。それに対し、並列 BCJR アルゴリズムは漸化式を展開しただけであるため、性能の劣化はない。しかし、分割 BCJR アルゴリズムは性能劣化を十分補うことができる、処理遅延の削減が可能である。2 分割すれば単純に処理遅延は 1 ／ 2 となり、4 分割すれば 1 ／ 4 となる。よってこの分割 BCJR アルゴリズムと並列 BCJR アルゴリズムを併用すれば、更なる処理遅延の削減が可能となる。今後この分割 BCJR アルゴリズムについて検討する必要がある。

謝辞

本研究に於いて懇切なる御指導、御鞭撻を賜った岩田 誠 教授に心より感謝の意を表します。

本研究の基礎としているデータ駆動型アーキテクチャを提唱された寺田 浩詔 教授に心より感謝の意を表します。

本研究に於いて、副査をお受け頂き、様々なご助言を賜った、福本 昌弘 助教授、
浜村 昌則 講師 に心より感謝の意を表します。

本研究を進めるにあたり、適切なる御助言、御指導を賜った 大森 洋一助手 に深く感謝の意を表します。

ご多忙の中、本研究の指導に貴重な時間を費やして頂いた、林 秀樹 氏、森川 大智 氏、
志摩 浩 氏に心より感謝の意を表します。

日頃から温かい御支援、並びに御助言を頂いた大学院生の別役 宣奉 氏、橋本 正和 氏に心より感謝の意を表します。

日頃から、多くの御意見、御支援を頂いた大学院生の、小倉 通寛 氏、三宮 秀次 氏、
中村 勲二 氏に、感謝の意を表します。

日頃から御意見、御支援を頂いた岩田研究室の方々、岩井 秀樹 氏をはじめ、
荒木 俊介 氏、大石 祐子 氏、　田 紘貴 氏、西山 直人 氏、山岡 正明 氏に感謝の意を表します。

日頃から御意見、御支援を頂いた岩田研究室の後輩の方々、朝日山 輝久 氏、
小笠原 新二 氏、白根 裕太 氏、千頭 裕子 氏、濱田 康裕 氏、山本 真弘 氏に感謝の意を表します。

参考文献

[1] B.Vucetic and J.Yuan, “ TURBO CODES: Principles and Applications, ” Kluwer Academic Publishers,2000

[2] H.Terada, et al,“ DDMP’s: self-timed super-pipelined data-driven multimedia processors, ” Proc.of the IEEE,87(2),pp.282-296(1999)

[3] 松本 渉, 落合 秀樹, “ OFDM 変調方式の応用, ” トリケップス,pp.63-85,2001

[4] 越智 俊輔, “ MAP アルゴリズムの並列処理による Turbo-Code の高速復号法, ” 東京工業大学修士論文,1998

[5] 松尾 憲一, “ スペクトラム拡散技術のすべて-CDMA から IMT-2000,Bluetooth まで-, ” 東京電機大学出版局,2002

[6] 萩原 春生, 大橋 正良, “ ターボ符号-連接符号化繰り返し復号-, ” 電子情報通信学会誌,2001

[7] 今井 秀樹, “ 符号理論, ” 電子情報通信学会,1990