RC-011

一般化 BP アルゴリズムを用いた LDPC 符号復号回路の設計と評価

Design and Evaluation of LDPC codes decoder with Generalized Belief Propagation

橋本　顕義 †
Akiyoshi Hashimoto

森野　東海 †
Harumi Morino

概要

復号回路に実装できる一般化 BP(Belief Propagation) アルゴリズムを提案し，これを最適に実行できる復号回路の方式を提案した。さらにこの方式を実装した復号回路を設計した。そして，訂正限界近傍での復号時間を測定し，Sum-Product 法の約 30% に削減されたことを確認した。これはスループットに換算すると Sum-Product 法の約 3 倍のスループットを達成したことを意味する。最後に回路規模 (ロジック部だけ) を見積もり，Sum-Product 法の 60% 増という結果を得た。

1. はじめに

低密度パリティ検査符号 (LDPC: Low-Density Parity Check) は 1960 年代に Gallager によって発見され [1] [2]，Mackay, Neal によって 1990 年代に再発見された [3] [4] 誤り訂正符号である。符号長が大きい極限で BCH 符号，Reed-Solomon 符号と比較して優れた訂正能力を持つため，近年，ネットワーク，光ディスク，ハードディスク，SSD(Solid State Drive) など多くの用途で利用されている。

LDPC 符号の標準的な復号アルゴリズムは Sum-Product 法 [5] である。1990 年代末に Sum-Product 法と統計力学における Bethe 近似 [6] [7] との類似が指摘され [8] [9]，統計力学の手法を用いた研究がなされてきた [10]。

Yedidia はさらに近似の精度を高めたアルゴリズムを提案し，これを一般化 BP (Genelarized Belief Propagation) アルゴリズムと呼んだ [11]。しかし，Yedidia のアルゴリズムは復号回路 (ハードウェア) を考慮したものとはいえなかった。

そこで，本研究では，復号回路に実装できる一般化 BP アルゴリズムを提案し，実際に一般化 BP アルゴリズムを実装した復号回路を設計し，さらにその性能を評価し，一般化 BP アルゴリズムの効果を明らかにすることを目的とする。 本論文の構成は以下の通りである。2 節では Bethe 近似と一般化 BP アルゴリズムについて説明する。その上で本研究のアルゴリズムを説明する。3 節では，本研究の復号回路の詳細について説明する。4 節では本研究の復号回路の性能評価結果を説明する。最後に 5 節にて本論文の結論を述べる。

† (株) 日立製作所 研究開発グループ 情報通信イノベーションセンタ

2. 一般化 BP アルゴリズム

本節では，復号回路に適用可能な一般化 BP アルゴリズムを定式化する。本節での説明は簡明な説明につとめたため，詳細は [7] [10] [11] などを参照されたい。

まず，本研究のアルゴリズムの基本的なアイデアを，磁性体の代表的な模型であるイジング模型 [7] を使って説明する。図 1 にイジング模型の模式図を示す。イジング模型は粒子が格子上に配置され，それぞれの粒子は磁気を帯びている。図 1 では粒子につけられた矢印が磁気の方向を示している。そして，隣接する粒子が相互作用をする。すべての粒子の帯びる磁気の向きが揃ったとき，この多体系は磁気を帯びる。粒子の帯びる磁気の向きがバラバラだと，この多体系は磁気を帯びない。イジング模型では隣接粒子の相互作用のみを考慮する。図 1 の中央の粒子 A は隣接する粒子 B，C，D，E の影響を受けている。粒子 B,C,D,E もまた隣接粒子の影響を受けているので，粒子 A の挙動を決定することは不可能である。しかし，粒子 B，C，D，E は他の粒子から影響をうけないと仮定して粒子 A の挙動を決定するのが，Bethe 近似である。

これに対して，注目する粒子の隣接ノードのさらに隣接するノードからの影響を考慮した，さらに高精度な近似法が考えられる。 これが Yedidia が提案した一般化 BP アルゴリズムである [11]。一般化 BP アルゴリズムの模式図を図 2 に示す。注目している粒子 A は，隣接する粒子 C の影響を受けているが，粒子 C は粒子 F，G，H の影響を受けている。一般化 BP アルゴリズムは，粒子 F，G，H までの影響を考慮した近似なのである。言い換えると Bethe 近似は 1 ホップまでの影響を考慮する近似であるのに対して，一般化 BP アルゴリズムは 2 ホップまでの影響を考慮する近似と言える。

本研究のアルゴリズムを述べる前に，Sum-Product 法について概説する。詳細は [5] [12] を参照されたい。送信語を $\boldsymbol{x}$(N 次元ベクトル)，受信語を $\boldsymbol{y}$(N 次元ベクトル) とする。$H = (H_{ij})$ を $M \times N$ パリティ検査行列とする。インデックスの集合 $A(m,n)$ を

$$A(m,n) \triangleq \{j \in \mathbb{N} | H_{mj} = 1 \text{ and } j \neq n\} \qquad (1)$$

と定義する。さらに n が省略されたとき

$$A(m) \triangleq \{j \in \mathbb{N} | H_{mj} = 1\} \qquad (2)$$

とする。 同様にインデックスの集合 $B(n,m)$ を

$$B(n,m) = \{i \in \mathbb{N} | H_{in} = 1 \text{ and } i \neq m\}, \qquad (3)$$

と定義する。さらに

$$B(n) = \{i \in \mathbb{N} | H_{in} = 1\} \qquad (4)$$

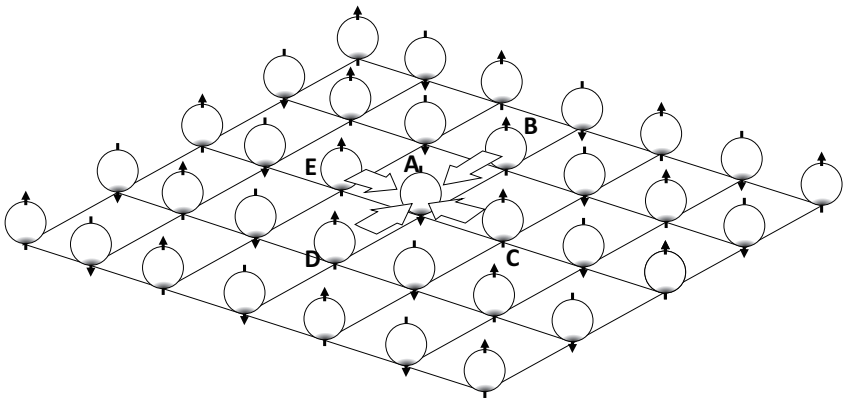

図 1　イジング模型の模式図

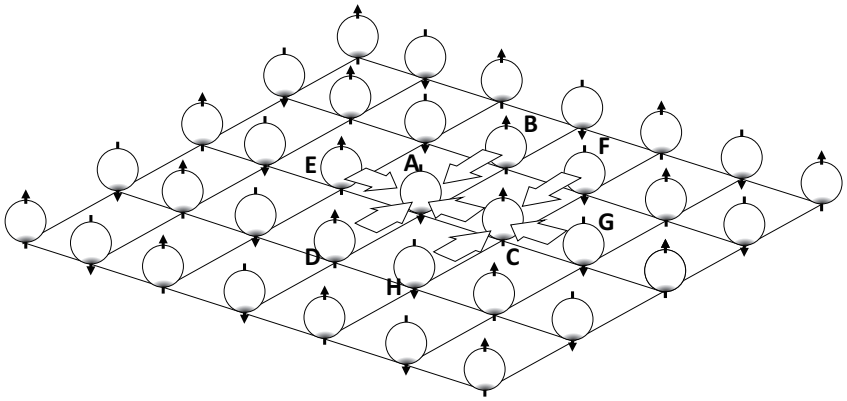

図 2　一般化 BP アルゴリズムの模式図

を定義する。Sum-Product 法は Bayes の定理を用いて，受信語 $\boldsymbol{y}$ から送信語 $\boldsymbol{x}$ の事後確率を計算してその値を推定する近似解法である。条件付き確率の公式

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

をこの問題に適用すると

$$P((x_n = 0)|\boldsymbol{y}) = \frac{P((x_n = 0) \cap \boldsymbol{y})}{P(\boldsymbol{y})}$$

となる。ここで，y_n とその他の $\boldsymbol{y}$ の成分を分離する。

$$\begin{aligned} P((x_n = 0)|\boldsymbol{y}) &= \frac{P((x_n = 0) \cap y_n \cap \{y_i\}_{i \in A(n)})}{P(y_n \cap \{y_i\}_{i \in A(n)})} \\ &= P(y_n|x_n = 0) P((x_n = 0)|\{y_i\}_{i \in A(n)}) \end{aligned}$$

同様に

$$P((x_n = 1)|\boldsymbol{y}) = P(y_n|x_n = 1) P(x_n = 1|\{y_i\}_{i \in A(n)})$$

を得る。x_n の事後対数尤度比 $\lambda(x_n|\boldsymbol{y})$ は

$$\begin{aligned} \lambda(x_n|\boldsymbol{y}) &= \log \frac{P((x_n = 0)|\boldsymbol{y})}{P((x_n = 1)|\boldsymbol{y})} \\ &= \log \frac{P(y_n|(x_n = 0))}{P(y_n|(x_n = 1))} + \log \frac{P((x_n = 0)|\{y_i\}_{i \in A(n)})}{P((x_n = 1)|\{y_i\}_{i \in A(n)})} \end{aligned}$$

となる。この第 1 項はビット n に関する事前対数尤度比である。これを既知と仮定する。そしてこれを λ_n とすると

$$\lambda(x_n|\boldsymbol{y}) = \lambda_n + \log \frac{P((x_n = 0)|\{y_i\}_{i \in A(n)})}{P((x_n = 1)|\{y_i\}_{i \in A(n)})} \qquad (5)$$

となる。ここで，チェックビット z_{mn} を以下のように定義する。

$$z_{mn} \triangleq \sum_{j \in A(m,n)} x_j$$

ここで，和は有限体 $\mathbb{F}_2$ の加法 (xor) である。このように z_{mn} を定義すれば，

$$z_{mn} + x_n = 0$$

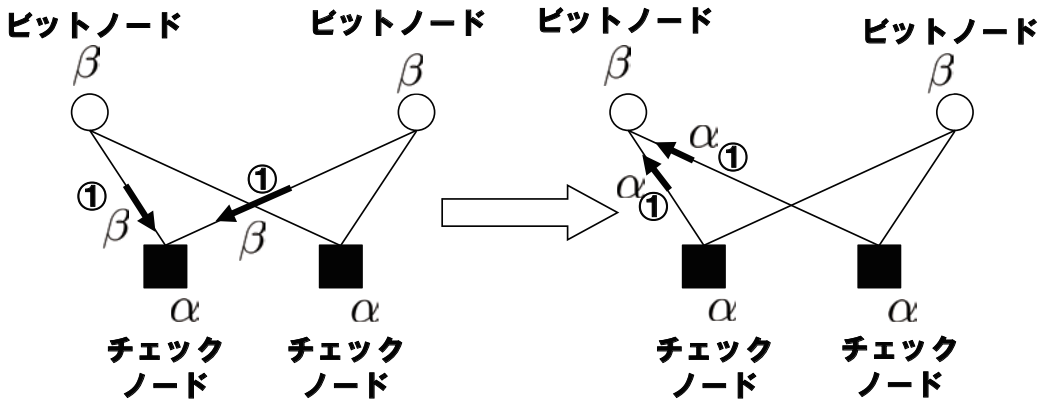

図 3　行処理 (左) と列処理 (右)

となる。$z_{mn}=1$ ならば $x_n=1$, $z_{mn}=0$ ならば $x_n=0$ である。よって，(式 5) は

$$\lambda(x_n|\boldsymbol{y})=\lambda_n+\log\frac{P((z_{mn}=0,\forall m\in B(n))|\{y_i\}_{i\in A(n)})}{P((z_{mn}=1,\forall m\in B(n))|\{y_i\}_{i\in A(n)})}$$

となる。$P(z_{mn}=0,\forall m\in B(n)|\{y_i\}_{i\in A(n)})$ は任意の $m\in B(n)$ に関する確率なので，それぞれの $m\in B(n)$ に関する確率の積でかける。すなわち，

$$P((z_{mn}=0,\forall m\in B(n))|\{y_i\}_{i\in A(n)})=\prod_{m\in B(n)}P((z_{mn}=0)|\{y_i\}_{i\in A(n)})$$

である。したがって，

$$\begin{aligned}\lambda(x_n|\boldsymbol{y})&=\lambda_n+\log\frac{\prod_{m\in B(n)}P((z_{mn}=0)|\{y_i\}_{i\in A(n)})}{\prod_{m\in B(n)}P((z_{mn}=1)|\{y_i\}_{i\in A(n)})}\\&=\lambda_n+\sum_{m\in B(n)}\log\frac{P((z_{mn}=0)|\{y_i\}_{i\in A(n)})}{P((z_{mn}=1)|\{y_i\}_{i\in A(n)})}\end{aligned}\quad(6)$$

を得る。$\lambda(x_n|\boldsymbol{y})\geqq 0$ ならば $x_n=0, \lambda(x_n|\boldsymbol{y})<0$ ならば $x_n=1$ と推定できる。しかし，式 (6) の第二項に事後確率が残っていてそのままでは左辺を計算できず，x_n を推定できない。ここで，チェックビット z_{mn} に対する対数尤度比を α_{mn}, 送信語の第 n ビットに対する事後対数尤度比を β_{mn} とおく。詳細は省略するが，α_{mn} と β_{mn} の間には

$$\alpha_{mn}=\left(\prod_{n'\in A(m,n)}\mathrm{sign}(\beta_{mn'})\right)f\left(\sum_{n'\in A(m,n)}f(|\beta_{mn'}|)\right)\quad(7)$$

という関係がある。ここで関数 f は

$$f(x)=\log\frac{e^x+1}{e^x-1}\quad(8)$$

である。さらに，(式 6) は

$$\beta_{mn}=\lambda_n+\sum_{m'\in B(n,m)}\alpha_{m'n}\quad(9)$$

と変形できる。α_{mn} と β_{mn} はどちらかが決まらなければ決定できない関係にあり，厳密に α_{mn} と β_{mn} を求めることはできない。これはイジング模型ですべての粒子の影響を考慮した解を求めることができないということに対応している。しかし，$\beta_{mn}=\lambda_n$ とおいて，(式 7) を実行し，得られた α_{mn} を (式 9) に代入すると新たな β_{mn} が得られる。このように行処理 (式 7) 列処理 (式 9) を交互に実行して事後対数尤度比 α_{mn},β_{mn} を更新していき，求めたい事後対数尤度比

$$\lambda(x_n|\boldsymbol{y})=\lambda_n+\sum_{m\in B(n)}\alpha_{mn}$$

が正負どちらかに発散していけば，対応するビット x_n が 0 か 1 かを高い確率で推定できることになる。このように Sum-Product 法は繰り返し計算による近似解法であるといえる。

行処理 (式 7) は二部グラフ (図 3 参照) において，注目するチェックノードに直接接続したビットノードからのメッセージを集約する処理，列処理 (式 9) は二部グラフにおいて注目するビットノードに直接接続したチェックノードからのメッセージを集約する処理である。いずれもグラフの 1 ホップまでの影響を考慮した処理となっており，Bethe 近似に対応している。図中の”①” がホップ数を示しており，行処理は注目したチェックノードに直接接続されたビットノードからのメッセージを集約しており，1 ホップである。列処理についても，注目したビットノードに直接接続されたチェックノードからのメッセージを集約しており，1 ホップである。

このように Sum-Product 法をみたときに，一般化 BP アルゴリズムに対応する 2 ホップまでの寄与を考慮した処理を検討する。 Sum-Product 法では行処理で注目するチェックノードの α_{mn} を更新する場合，二部グラフで当該チェックノードに接続されたビットノードの β_{mn} を使う。これは 1 ホップの処理に対応する。

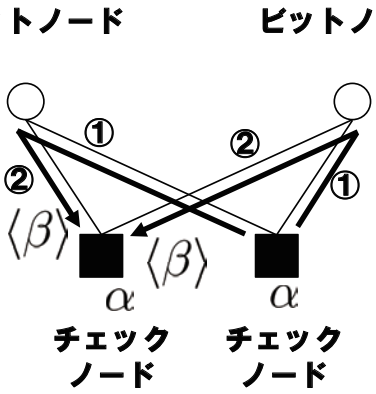

図 4　本研究のアルゴリズムの説明図

そこで，前記ビットノードそれぞれについて，二部グラフにおいて接続されたチェックノードの α_{mn} からのメッセージを集約する。集約した結果を $\langle\beta\rangle$ とする。α_{mn} の更新にこの $\langle\beta\rangle$ を使うと 2 ホップの寄与を取り入れたことになる。これを二部グラフで表現した図が図 4 である。図 3 と同様に ◯ で囲まれた数字がホップ数である。チェックノードからビットノードにメッセージを集約する処理で 1 ホップ，ビットノードで集約されたメッセージを更に集約する処理で 1 ホップと，合計 2 ホップの寄与を取り入れている。

チェックノードからビットノードにメッセージを集約する処理は，列処理 (式 9) である。そこで，あるチェックノードの事後対数尤度比 α_{mn} を更新する際に，前記チェックノードと直接接続されたビットノードそれぞれについて，列処理 (式 9) を実行し，その結果を行処理 (式 7) に代入することが，事後対数尤度比 α_{mn} を更新する際に，2 ホップまでの寄与を考慮した復号法に対応する。

これまでの考察をまとめると以下のアルゴリズムを得る。

定理 1 (一般化 BP アルゴリズム)
step 0:
任意の $m \in A(n), n \in B(m)$ に対して，$\alpha_{mn} = 0$ とする。反復回数上限値を $l_{\max}$ とし，反復回数 l を 1 とする。
step 1:行処理
$m = 1, 2, \cdots, M$ に対して

$$\alpha_{mn} = \left(\prod_{n' \in A(m,n)} \mathrm{sign}(\langle\beta_{mn'}\rangle) \right) f\left(\sum_{n' \in A(m,n)} f(|\langle\beta_{mn'}\rangle|) \right) \tag{10}$$

を計算する。ただし

$$\langle\beta_{mn'}\rangle = \lambda_{n'} + \sum_{i \in B(n',m)} \alpha_{in'} \tag{11}$$

である。
step 2: 一時推定語の計算
本アルゴリズムが一時的に推定した符号語である一時推定語 $\hat{\boldsymbol{c}}$ を

$$\hat{c}_n = \begin{cases} 0 & (\mathrm{sign}(\lambda_n + \sum_{i \in B(n)} \alpha_{in}) = 1) \\ 1 & (\mathrm{sign}(\lambda_n + \sum_{i \in B(n)} \alpha_{in}) = -1) \end{cases}$$

という条件で計算する。
step 3: パリティチェック
一時推定語が

$$H\hat{\boldsymbol{c}} = \boldsymbol{0} \tag{12}$$

を満たしているかチェックする。
step 4: パリティチェック判定
もし $H\hat{\boldsymbol{c}} = \boldsymbol{0}$ ならば $\hat{\boldsymbol{c}}$ を出力し，そうでなければ $l = l+1$ とし，ステップ 1 に行く。

ただし，関数 f をハード的に求めることは難しいので，回路実装では以下のように近似する [12]。

$$\alpha_{mn} = \kappa \left(\prod_{n' \in A(m,n)} \mathrm{sign}(\langle\beta_{mn'}\rangle) \right) \min_{n' \in A(m,n)} |\langle\beta_{mn'}\rangle| \tag{13}$$

κ は訂正能力が最大になるように最適に調整されるパラメータである。

ビットノードの事後対数尤度比 β_{mn} は不要になり，一時的な変数 $\langle\beta_{mn}\rangle$ のみが必要になる。また，列処理がなくなり，行処理のみになっている。しかし，行処理では更新対象となっているチェックノードに接続されたビットノードそれぞれについて列処理相当の計算 (式 11) が必要になる。したがって，Sum-Product 法と比較して，計算量は増大していることに注意が必要である。しかし，Sum-Product 法と比較して，高い精度で対数尤度比を計算していることから，Yedidia が指摘しているように反復回数の削減が期待される [11]。

3. 復号回路の設計

本節では，一般化 BP アルゴリズムを実装した復号回路を説明する。本研究では，(10775, 8620) 擬似巡回型符号を採用した [13]。そのパリティ検査行列は 125 個の

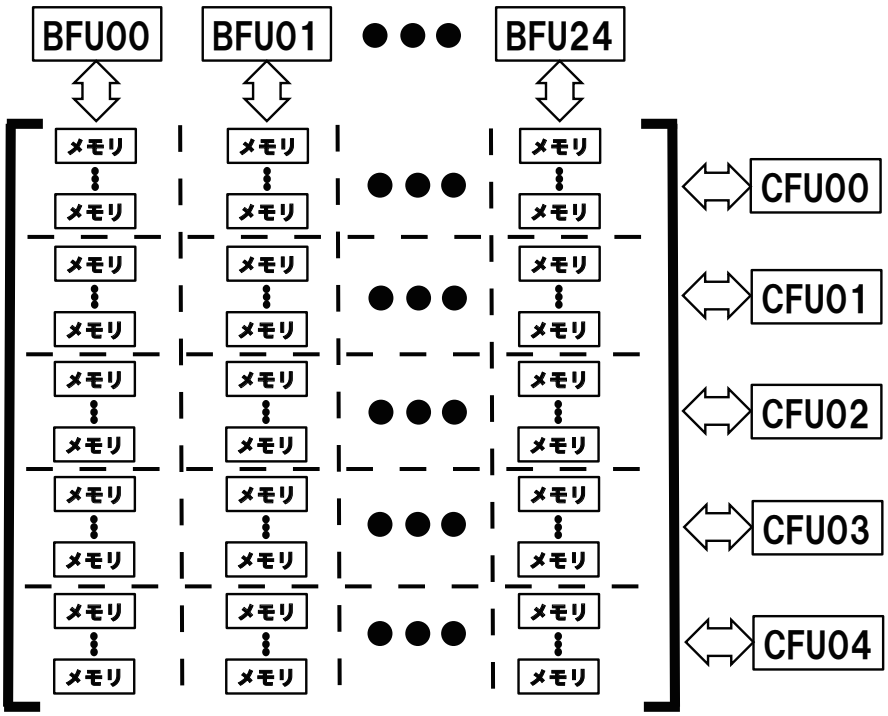

図 5　BFU, CFU とパリティ検査行列の関係

431×431 の部分行列からなる。そして，この部分行列が行方向に 5 個，列方向に 25 個並んだ構成をとる。行列の 1 行の中の 1 の数を” 行重み”，1 列の中の 1 の数を列重みという。この部分行列の行重み，列重みはそれぞれ 1 であり，巡回行列である。したがって，全体としては行重み 25，列重み 5 である。

本研究の復号回路を説明する前に従来の Sum-Product 法の復号回路について説明する。Sum-Product 法の復号回路とパリティ検査行列の対応関係を図 5 に示す。BFU(Bit Function Unit) は列処理を行うユニットで，部分行列に対応して 25 個搭載され担当する部分行列の列処理を実行する。CFU(Check Function Unit) は行処理を行うユニットで部分行列に対応して，5 個搭載され，対応する部分行列内の行処理を実行する。中央にあるメモリ群であるが，パリティ検査行列の行列要素が 1 の部位に対応してメモリが存在し，事後対数尤度比 α_{mn} または β_{mn} が格納される。行処理 (式 7) は 5 並列，列処理 (式 9) は 25 並列で実行される。

CFU の動作の概略を図 6 に示す。行処理 (式 7) において，CFU がパリティ検査行列の行を 1 つ選択すると，行上に存在するメモリが特定される (この場合 25 個)。図 6 に示すように CFU は前記特定されたメモリから β_{mn} を読みだして，(式 7) に従って α_{mn} を計算し，この結果を β_{mn} が格納されていたメモリに書き戻す。列処理 (式 9) についても同様で，選択されたパリティ検査行列の列に対応するメモリ (この場合 5 個) から α_{mn} を読み出し，(式 9) に従って β_{mn} を計算し，この結果を α_{mn} に格納されていた前記メモリに書き戻す。このように，α_{mn} と β_{mn} を同じメモリに格納するため，行処理 (式 7) と列処理 (式 9) は同時に実行できない。

一般化 BP アルゴリズムを回路化するためには，行処理 (式 10) と (式 11) について詳細に検討しなければならない。行処理を実行するわけであるから，復号回路がパ

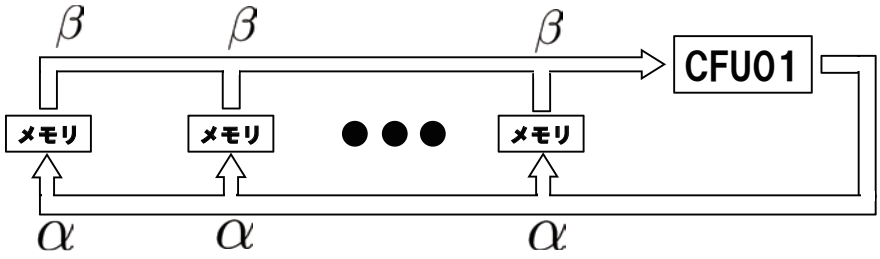

図 6　CFU の動作の概略

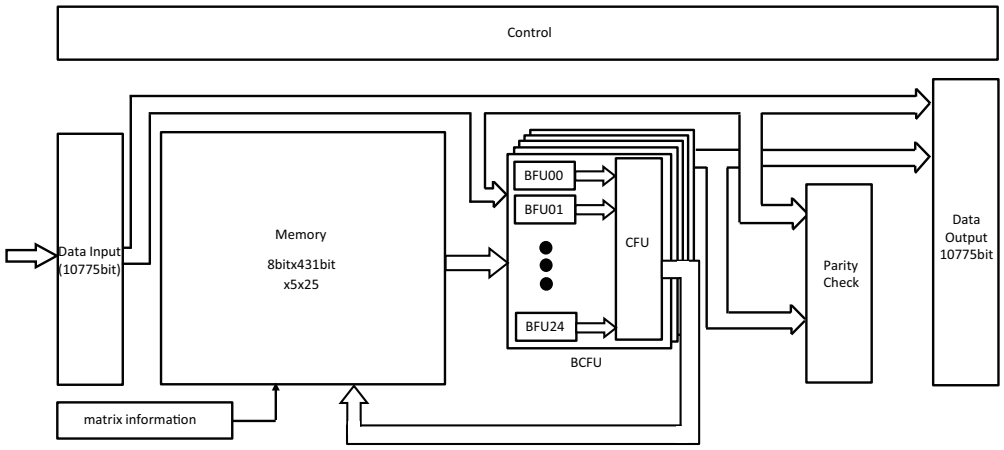

図 7　本研究の復号回路のブロック図

リティ検査行列の行を 1 つ選択すると，25 個の α_{mn} が選択される。その中から 1 つを選択して，(式 10) を実行するわけだが，そのためには，残り 25 個の α_{mn} に関して，(式 11) の計算をしなくてはならない。これをシリアルに実行していては，復号時間が大幅に伸びてしまう。そこで，(式 11) が Sum-Product 法の列処理 (式 9) と等価であることを利用して，1 つの CFU に 25 個の BFU を接続し，25 回の (式 11) を同時に実行してしまえばよい。そうすれば復号時間の増大は防止できる。

以上の議論から，本研究では図 7 のような回路構成をとった。入力データを一時的に格納する Data Input 部，事後対数尤度比を格納する Memory，演算を行う BCFU，パリティチェックを行う ParityCheck 部，出力データを一時保存する，Data Output 部である。また，パリティ検査行列の 1 の要素を記憶する Matrix Information 部と全体を制御する Control 部である。本研究の復号回路の特徴は BCFU にある。BCFU は 5 つあり，それぞれ，行方向の部分行列に対応する。そして，それぞれの BCFU は上記の議論から 25 個の BFU と 1 個の CFU からなる。BFU00 ∼BFU24 は Memory から α_{mn} を読み込み，(式 11) を実行し，演算結果 $\langle\beta_{mn}\rangle$ を CFU に送り込む。CFU は 25 個の $\langle\beta_{mn}\rangle$ を受け取り，(式 10) を実行する。そして結果 α_{mn} を Memory に書き戻す。この方式であれば，(式 11) に起因する計算量の増大が復号時間の増大に繋がらないと期待される。

図 8 に BFU のブロック図を示す。本研究の BFU は 2 段パイプライン構成となっており，複数の $\alpha_{mn'}$ と λ_n を加算している。BFU はパリティ検査行列の 1 つの列を選択すると，当該列上のすべての α_{mn} をメモリから同時に読み込んでくるため，パイプラインの前後でデータの依存性はない。よって,1 クロックごとに新たな $\langle\beta_{mn}\rangle$ を出力できる。

CFU を図 9 に示す。本研究の CFU は 4 段パイプライン構成となっている。まず，各々の $|\langle\beta_{mn}\rangle|$ の絶対値を計算し，$|\langle\beta_{mn}\rangle|$ の中から最小値を求める。そして，κ を乗じて，最後に $\langle\beta_{mn}\rangle$ の符号の積 (最上位ビットの xor 結果) をかけて α_{mn} を得る。CFU はこの回路を行重みだけ持っており (本研究の場合 25)，パリティ検査行列の 1 行の中に存在する α_{mn} をすべて同時に更新できる。すなわち，パイプラインの前後でデータの依存関係はない。さらに，Sum-Product 法も一般化 BP アルゴリズムも異

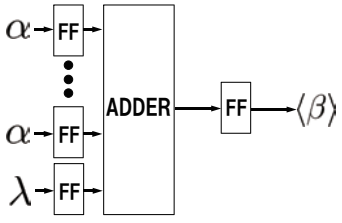

図 8　BFU のブロック図

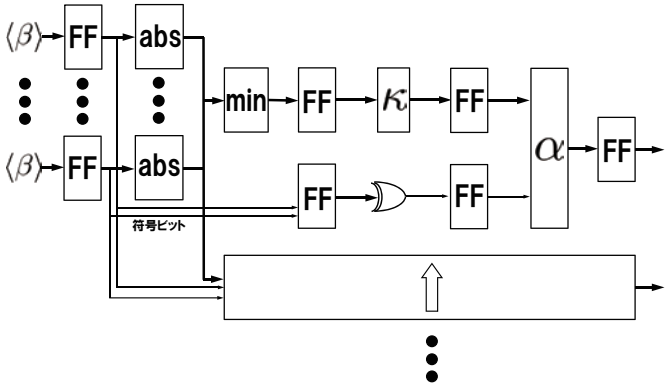

図 9　CFU のブロック図

なる行の間にデータの依存関係はない。そして，CFU はパイプライン構成をとっているため，1 クロックごとに新たな α_{mn} を Memory に出力できる。

BFU も CFU もパイプライン構成をとっているので 1 クロックごとに演算結果を出力する。これらを直結した BCFU は 6 段パイプラインの演算器となり，パイプラインレイテンシは 6 クロックであるものの，1 クロックごとに新たな α_{mn} を出力できる。Sum-Product 法の復号回路は列処理が完了した後に行処理を実行していた。行処理と列処理は同時に実行できなかった。本研究の復号回路はパイプラインのために事実上行処理しか実行しない。また，行処理と列処理はほぼ同じ時間がかかるため，本研究の復号回路のスループットは Sum-Product 法と比較して原理的に 2 倍となる。

4. 評価結果

4.1 復号時間評価結果

本研究では Cadence®NCsim 回路シミュレータにて本研究の復号回路の復号時間を評価した。性能評価条件を表 1 に示す。Sum-Product 法の課題であるエラービットが多くなると，反復回数が多くなり，結果として復号時間が長くなるという課題に対して提案方式がどれくらい効果があるかを検証するため，エラービット数を訂正限界近傍の 204 ビット固定とした。

表 1　性能評価条件

通信路	BSC
符号語数	100
最大反復回数	250
データ	ランダムデータ
エラービット数	204 ビット固定
エラービット分布	一様分布
入力クロック周波数	100MHz

性能測定結果を表 2 に示す。一般化 BP アルゴリズムの復号回路の反復回数は Sum-Product 法のそれと比較して約 60% に低減している。一般化 BP アルゴリズムの復号回路の平均復号時間は，Sum-Product 法のそれと比較して，約 30% になっている。スループットに換算すると 3 倍の改善である。この点について検討するため，両者の動作を比較した。両者を比較したシミュレーション波形画像を図 10 に示す。上段が Sum-Product 法の波形，下段が一般化 BP の波形である。両者とも同一データ，同一エラーパターンという条件下での動作である。Sum-Product 法では行処理 → 列処理で 1 回の反復となる。図 10 では 19 回反復計算して訂正に成功している。一方，一般化 BP アルゴリズムでは行処理だけで 1 回の反復となり，11 回反復計算して訂正に成功している。たしかに，反復回数は約 60% に低減している。ここに一般化 BP アルゴリズムの効果が確認できたといえる。また Sum-Product 法では訂正に 170,000ns かかっているのに対して，一般化 BP アルゴリズムでは訂正に 60,000ns かかっている。確かに訂正時間は約 30% に減少している。この理由はおおむね以下の通りである。Sum-Product 法では行処理に 431 クロック (= 部分行列のサイズ)，列処理に 431 クロックかかり，1 回の反復で 862 クロックかかる。一方，一般化 BP アルゴリズムでは，行処理だけになるため，431 クロックになることが予想される。これは図 7 のような構成をとり，BFU を 5 倍に増やすことで，増大した計算量に対応した結果である。しかし，実際の回路設計上の問題 (パイプラインの段数が増えたことによるレイテンシ増大，独立ユニットが減少したことにより，並行動作できる余地が少なくなった点) で 1 回の反復で 431 + 51 クロックかかった。また，データ入力，出力といった改造していない部位の処理時間も無視できなかった。よって，アーキテクチャ改良による効果は 2 倍に到達できなかった。このような事情で，反復回数減少

表 2　性能測定結果

	Sum-Product 法	一般化 BP
平均反復回数	31.4	18.4
平均復号時間	2.83×10^5ns	9.72×10^4ns

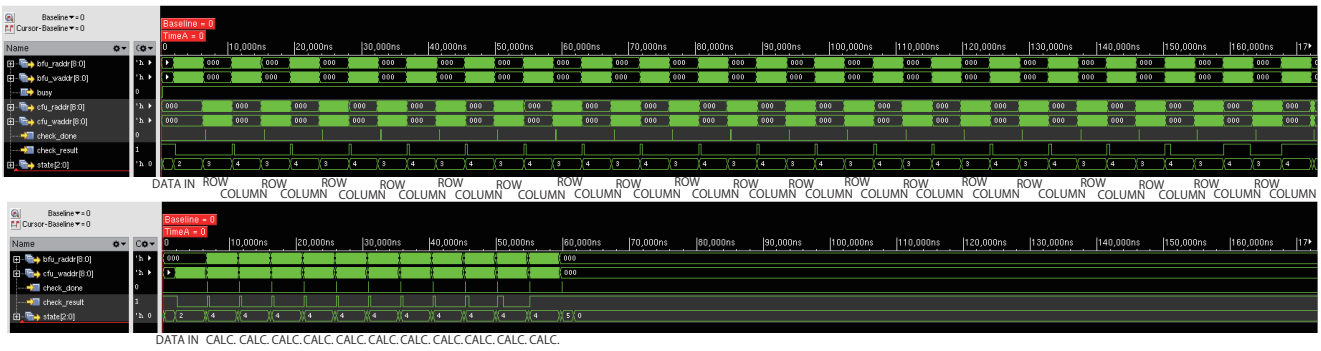

図 10　波形の比較

表 3　論理合成条件

条件	内容
プロセス	TSMC 40nm
クロック周波数	200MHz

表 4　論理合成結果 (単位:gate)

	Sum-Product 法	一般化 BP
BFU+CFU 合計	135721	269691
Input	133552	267398
Output	134523	138825
Parity Check	71516	70480
Cotrol	1319	1601
memwrap	43791	70437
register	1053	1053
total	521475	819486

表 5　Memory 要求構成

	Sum-Product 法	一般化 BP
SRAM	8bit×431 ワード 1 リードポート 1 ライトポート	8bit×431 ワード 5 リードポート 1 ライトポート
個数	5×25	5×25

で約 2 倍，アーキテクチャ改善で約 2 倍で合計 4 倍のスループット向上が見込めるところ，3 倍のスループット向上になったと結論付けられる。

4.2　回路規模評価結果

Sum-Product 法と一般化 BP アルゴリズムの復号回路を論理合成して回路規模を比較した。論理合成条件を表 3 に示す。ロジック部の論理合成結果を表 4 に示す。Memory については，合成環境がないため，必要な構成を表 5 に示す。一般化 BP アルゴリズムによる計算量の増大を BFU を増やすことで対応した結果，BFU+CFU の回路規模が約 2 倍に増大している。また，BFU が増えて同時に Memory にアクセスさせる必要がでてきたため，1 個のメモリに 5 つのリードポートをつけることになった。そのため，Memory の周辺論理である mempwrap 部の回路規模も約 2 倍に増大することになった。そのため，ロジック部の回路規模は 1.6 倍となった。

5.　結論

復号回路に実装できる一般化 BP アルゴリズムを提案し，これを最適に実行できる復号回路の方式を提案した。さらにこの方式を実装した復号回路を設計した。そして，訂正限界近傍での復号時間を測定し，Sum-Product 法の約 30% に削減されたことを確認した。これはスループットに換算すると Sum-Product 法の約 3 倍のスループットを達成したことを意味する。最後に回路規模 (ロジック部だけ) を見積もり，Sum-Product 法の 60% 増という結果を得た。

参考文献

[1] R. Gallager, “Low-density parity-check codes,” Information Theory, IRE Transactions on, vol.8, no.1, pp.21–28, jan. 1962.
[2] R.G. Gallager, Low-Density Parity Check Codes, THE MIT PRESS, 1963.
[3] D.J.C. MacKay and R.M. Neal, “Near shannon limit performance of low density parity check codes,” Electronics Letters, vol.33, no.6, pp.457–458, March 1997.
[4] D.J.C. MacKay, “Good error-correcting codes based on very sparse matrices,” Information Theory, IEEE Transactions on, vol.45, no.2, pp.399–431, March 1999.
[5] T.K. Moon, Error Correction Coding: Mathematical Methods and Algorithms, Wiley-Interscience, 2005.
[6] H.A. Bethe, “Statistical theory of superlattices,” Proceedings of The Royal Society London A, vol.150, no.871, pp.552–575, July 1935.
[7] S.R.A. Salinas, Introduction to Statistical Physics, Springer, 2001.
[8] Y. Kabashima and D. Saad, “Belief propagation vs. tap for decoding corrupted messages,” EPL (Europhysics Letters), vol.44, no.5, p.668, 1998.
[9] Y. Kabashima and D. Saad, “Statistical mechanics of error-correcting codes,” EPL (Europhysics Letters), vol.45, no.1, p.97, 1999.
[10] M. Opper and D. Saad, eds., Advanced Mean Field Methods, The MIT Press, 2001.
[11] J.S. Yedidia, W.T. Freeman, and Y. Weiss, “Generalized belief propagation,” Technical Report of Mitsubishi Electric Research Laboratory, vol.2000, p.26, 2000.
[12] 和田山正，低密度パリティ検査符号とその復号法，トリケップス，2002.
[13] M.P.C. Fossorier, “Quasicyclic low-density parity-check codes from circulant permutation matrices,” Information Theory, IEEE Transactions on, vol.50, no.8, pp.1788–1793, Aug. 2004.