3ZB-5

時間的・空間的並列処理による算術符号化・復号の高速アーキテクチャの提案

The Proposition of the High-Speed Architecture for Arithmetic Encoding and Decoding by Time-Based and Space-Based Parallelization

若林 俊一 † 菅谷 至寛 † 阿曽 弘具 †
Shunichi Wakabayashi Yoshihiro Sugaya Hirotomo Aso

1. はじめに

近年の通信や記憶装置のスループットの向上は目覚しく，また情報化により扱われるデータ量の増大も著しい．伝送・記録を効率良く行うためデータ圧縮技術の重要性は依然として高い．これに対処する高スループットの符号化・復号を汎用プロセッサで実現するのは困難な場合があり，効率の良い専用ハードウェアが求められる．

本研究では算術符号を対象とし、時間的並列化と空間的並列化を併用した符号化･復号双方を共通のハードウェアで行える専用アーキテクチャを提案する．これにより圧縮率をほとんど損なわずに高いスループットでの処理を実現した．

2. 符号の概要

データ圧縮は，特定の情報源に特化した手法，不可逆な手法など用途に応じ様々な種類があるが，本研究では汎用性を重視した可逆圧縮を対象とした．可逆圧縮には文脈上の特徴を利用した辞書的手法，情報源の統計的確率に応じた符号語を扱う統計的手法などがある．ここでは統計的手法の一種である適応型算術符号を対象とする．これは記憶容量や計算量をあまり必要とせず，効率的な並列化が可能である．

2.1 算術符号

符号化及び復号を行う情報源アルファベット集合を $X = \{0, \ldots, 2^N - 1\}$，$X$ の生起確率ベクトルを $\boldsymbol{p} = \begin{bmatrix} p(0) & \ldots & p(2^N - 1) \end{bmatrix}^{\mathrm{T}}$，アルファベット系列を $x_0, x_1, \ldots, x_{L-1}$ とする．

符号化では，下に閉じた半開区間 $I_i = [l_i, h_i)$ より，

$$\emptyset \neq f(I, x, \boldsymbol{p}) \subset I$$
$$f(I, x, \boldsymbol{p}) \cap f(I, x', \boldsymbol{p}) = \emptyset \qquad (x \neq x')$$

の条件を満たす関数 f 及び $x_i, \boldsymbol{p}$ を用いて，新たな区間 $I_{i+1} = f(I_i, x_i, \boldsymbol{p})$ の算出を，$I_0 = [0, 1)$ から $i = 0, \ldots, L-1$ で繰り返し行う．最終的に得られる I_L に含まれる有限桁の実数を，系列全体を表す符号語 y として出力する．

復号では $i = 0$ から始めて，I_i に対して $y \in f(I_i, x_i, \boldsymbol{p})$ を満たす x_i を算出する。それより次の I_{i+1} を求め，以下同様に繰り返す。

$f(I, x, \boldsymbol{p})$ を I を各生起確率の比率で分割する関数と定め，y を最低限必要な桁数で表現することにより，算術符号化の圧縮率は Shannon のエントロピー限界とほぼ等しい圧縮率であることが知られている．

2.2 RangeCoder

算術符号の実装は困難なため，様々な近似手法が提案されている．本研究では RangeCoder [1, 2] をベースに並列アーキテクチャに適した改良を行った．

† 東北大学大学院 工学研究科

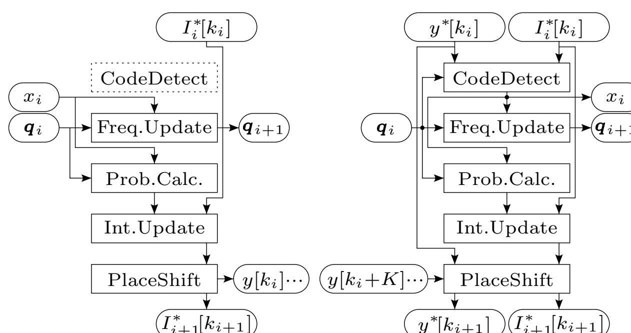

図 1: 符号化 (左)，復号 (右) における処理の概要

RangeCoder は I_i の上・下限値 h_i, l_i を b進小数と考え，桁の左シフトを繰り返すことにで非負整数の四則演算のみで近似を行う．符号の入出力は $\log_2 b$bit ずつまとめて行う。本研究では，符号器の空間的並列化を効率良く行うため，この特徴を利用した．

符号化では，上・下限値の小数第 k_i 位 $h_i[k_i], l_i[k_i]$ から K 桁分を抜き出した b進整数 $h_i^*[k_i], l_i^*[k_i]$ に対して x_i を用いて半開区間の更新演算を行う．得られた新たな上・下限値 $h_{i+1}^*[k_i], l_{i+1}^*[k_i]$ に対し，それらの上位で一致する桁数分シフトさせ，新たに抜き出す小数第 k_{i+1} 位を求める．区間の幅は $b^{k_{i+1}-k_i}$ 倍に拡大され，一致した $h_{i+1}[k_i], \ldots, h_{i+1}[k_{i+1} - 1]$ は $y[k_i], \ldots, y[k_{i+1} - 1]$ として出力する．$k_0 = 0$ とし，区間の分割と拡大を繰り返す．

復号では，i ステップ目で $l_{i+1}^*[k_i] \leq y^*[k_i] < h_{i+1}^*[k_i]$ が成り立つ x_i を決定し，その後符号化と同様の区間の更新を行う．$y[k_i + K], \ldots, y[k_{i+1} + K - 1]$ を入力から補い，同様の操作を繰り返す．x_0 を復号する前に初期状態 $y^*[0]$ を構成するため $y[0], \ldots, y[K-1]$ を入力する必要がある．

2.3 適応型符号における生起確率予測

生起確率ベクトル $\boldsymbol{p}$ が既知と仮定すること，または事前に全系列から算出することは，汎用･高スループットの用途に不適である．そこで，先行する系列 $x_0, \ldots, x_{i-1}$ から求まる頻度ベクトル $\boldsymbol{p}_i$ を次の x_i の処理で用いる生起確率とする。但し頻度計算に必要な各生起回数は，オーバーフロー防止及び生起確率の動的な変化に順応するため，周期的に定数で除算する．また常に $\boldsymbol{p}_i$ の各要素に対し $p_i(x) \neq 0$ が成り立つように考慮する必要がある．

3. アーキテクチャの概要

符号化・復号の処理の流れを図 1. に示す．$\boldsymbol{q}_i$ は生起回数を保持するための状態ベクトルである．符号化器は復号器の一部の機能を無効化し，切り替えることで共通化できる．生起回数の更新，生起確率の計算及び，復号時の二分探索により x_i の決定を行う部分は，後述の方

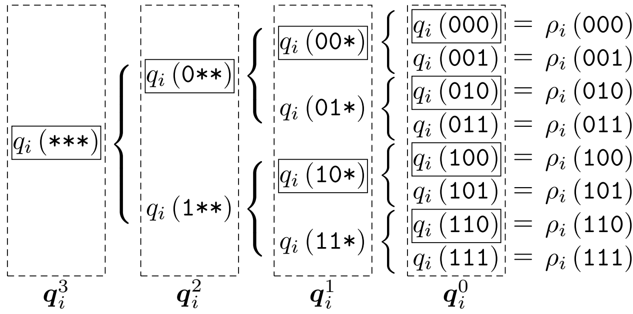

図 2: $N = 3$ における生起回数と内部表現の関係

法で一括処理することができる．さらに一連の処理を段階的に区切りパイプライン処理を行うことで，時間的な並列化を実現した．以降この部分を頻度処理器と呼ぶ．

半開区間の更新及び桁のシフトなど，頻度処理器以外はパイプライン化は困難であるが，空間的に同一の機能を複数配置し，同時に動作させることで空間的な並列化を実現した．以降この部分を符号器と呼ぶ．

3.1 頻度処理器おける時間的並列化

$x \in X$ を Nbit の 2 進値とする．I_{i+1} を求めるには，生起回数 $\rho_i(x_i)$ に対し，累積生起回数

$$\sigma_i(x_i - 1) = \sum_{x=0}^{x_i-1} \rho_i(x) \qquad \sigma_i(x_i) = \sum_{x=0}^{x_i} \rho_i(x)$$

を求める必要がある．この計算および生起回数の更新は，二分木状の関係を持つ内部状態ベクトル $\boldsymbol{q}_i$ を用いることで，$O(N)$ の計算量で行える [3]．図 2. のように x の bit の並びにより二分木を構成する．各節点は，配下にある $\rho_i(x)$ の総和を値として持つ．頻度処理器は，全ての兄弟関係の節点において図中上側の節点の値のみを状態として保持する．全状態数は 2^N で冗長性は無く，生起回数の更新は状態の更新で実現する．状態の関係より復号時の二分探索も容易である．

木の深さ $N - n$ の値を部分状態ベクトルを $\boldsymbol{q}_i^n$ $(0 \leq n \leq N)$ で表すと，全ての処理は，各部分状態ベクトルから，高々1 個ずつの要素の参照・更新で完了する．$\boldsymbol{q}_i^n$ に関わる処理を 1 つの Proccessing-Element, PE^n で行うと，$\mathrm{PE}^n \rightarrow \mathrm{PE}^{n-1}$ の依存関係のみが存在する．よって，頻度処理器は PE を用いて，図 3. のように $N + 1$ 段のパイプライン処理が可能である．

3.2 符号器における空間的並列化

図 1. より符号器の処理を待たないと頻度処理器は次の処理を開始できず，前述のパイプラインは機能しない．そこで M 個の符号器 $\mathrm{Coder}^0, \ldots, \mathrm{Coder}^{M-1}$ を空間的に複数配置し，x_i の処理を $\mathrm{Coder}^{i \bmod M}$ へと M 個毎に分配する．独立した M 個の半開区間と符号 $I_{i'}^0, \ldots, I_{i'}^{M-1}, y^0, \ldots, y^{M-1}$ を定義し，更新規則を

$$I_{\lfloor i/M \rfloor + 1}^{i \bmod M} = f\left(I_{\lfloor i/M \rfloor}^{i \bmod M}, x_i, \boldsymbol{p}_i\right)$$

とする．これにより頻度処理器は x_i の処理を $\mathrm{Coder}^{i \bmod M}$ が実行中に，$x_{i+1}, \ldots, x_{i+M-1}$ の処理をパイプラインを活かし同時に実行できる．また頻度処理器を共用しているため，符号器を並列化しても理論上圧縮率は変わらない．PE，符号器のクリティカルパス長を $t_{\mathrm{PE}}, t_{\mathrm{Coder}}$ とすると，$M \geq N + \lceil t_{\mathrm{Coder}} / t_{\mathrm{PE}} \rceil$ としたとき，スループットは t_{PE} のみで決定する．

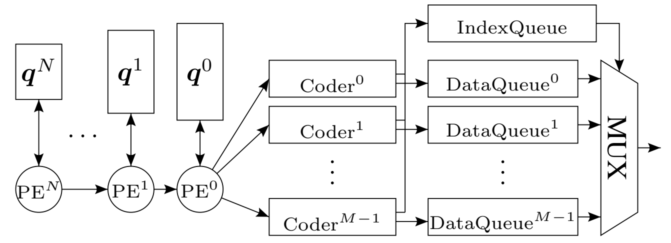

図 3: 頻度処理器，符号器の並列化と IndexQueue, DataQueue を用いた整列器

M 個の符号 $y^0, \ldots, y^{M-1}$ は，符号化では逐次単一の系列へとマージしながら出力し，復号では単一の系列から各符号器の入力へと分解を行う．各符号器の入出力するタイミングおよび符号長は不定である．符号化と復号において，入出力が行われるタイミングは等しいが K 桁分のずれがある．復号時に必要な情報が得られるように，符号化時に M 個の符号の適切なマージ処理が必要である．

マージ処理は各符号器からの出力タイミングを維持しつつ，全出力を K 桁ずらす整列器を用いて実現した．これは図 3. で示すように 1 個の IndexQueue と，M 個の DataQueue により構成される．Coder^m から出力が行われた場合，出力を $\mathrm{DataQueue}^m$ に，'m' を IndexQueue に追加する．IndexQueue の先頭が 'm' であったとき，$\mathrm{DataQueue}^m$ が要素を保持していれば，取り出して最終的な出力とする．各 DataQueue の初期状態は空，IndexQueue は $0, \ldots, M-1$ を K 個ずつ順に保持する．これは復号時に符号器が初期状態を構成するための入力を行う順序に相当する．

$$\mathrm{DataQueue}^m \text{の要素数} + K = \mathrm{IndexQueue} \text{内の} \text{ '}m\text{' } \text{の数}$$

が常に成立するため，復号時には各符号器の入力要求に応じて，マージされた符号系列からそのまま入力を行うだけで，適切に復号することが可能となる．

4. おわりに

頻度処理を行う PE のクリティカルパスは，整数加算器 2 個分程度の処理時間に相当する．すなわち，この間隔で 1 個のアルファベットの符号化および復号が行うことができ，提案手法によって高いスループットを実現することが可能であることが示された．

参考文献

[1] Michael Schindler, “A Fast Renormalisation for Arithmetic Coding,” IEEE Data Compression Conf., pp.572–, 1998

[2] Dmitry Subbotin, http://algo.4u.ru/, 1999

[3] Peter M. Fenwick, “A New Data Structure for Cmulative Frequency Tables,” Software–Practice & Experience, Vol.24(3), pp.327–336, Mar. 1994