ベクトル量子化のための並列コードブック生成アルゴリズムの性能評価

百瀬真太郎[†], 佐野健太郎[†], 滝沢寛之[‡],
中島平[§], Clecio Donizete Lima[†], 小林広明[‡], 中村維男[†]

[†]東北大学大学院情報科学研究科 [‡]東北大学情報シナジーセンター
[§]東北大学大学院工学研究科

ベクトル量子化は高効率なデータ圧縮手法であり、データの保存や転送において核となる技術である。これまでに、誤差の少ない量子化のための最適コードブックを生成する様々な手法が提案されており、中でもアルゴリズムの改良によってコードブック生成処理時間の短縮を図るLaw-of-the-Jungle(LOJ) アルゴリズムが注目を集めている。しかし、大きなデータセットを単一のCPUで処理する場合、アルゴリズムの改良による処理時間短縮には限界があり、並列処理によるさらなる速度向上が求められている。本論文では、メモリ分散型並列計算機に適した並列LOJアルゴリズムを提案する。IBM SP2、NEC AzusA、PCクラスタを用いて並列LOJアルゴリズムの性能評価を行なった結果、いずれもプロセッサ台数に対する高い速度向上率が得られた。

1 はじめに

近年、画像、動画、ボリュームデータ等の大規模データに対する劣化の少ない圧縮は、データの効率的な保存や転送を実現する上で、重要な技術となっている [1]–[7]。ベクトル量子化 (Vector Quantization, VQ)[8] は劣化の少ないデータ圧縮手法の1つで、スカラ量子化 (SQ) と比べ理論的に高い圧縮効率を実現する。

入力ベクトルを最も近いコードベクトルにより近似するVQでは、入力とこれを近似するベクトル間の距離により定義される歪みが最小となるような最適コードブックが求められる。これまでに、最適コードブック生成のための様々なアルゴリズムが提案されてきたが [8]–[11]、巨大なデータセットに対して適切なコードブックを生成するためには膨大な計算時間が必要となり、このことが実用化の際の大きな問題となっている。例えば、基本的な競合学習アルゴリズムであるコホネン学習 [12][13] は最適コードブックを得るまでに多くのコードベクトル更新回数を必要とする。またコードベクトル毎の利用頻度が異なり、特に未使用のコードベクトルが数多く残るという問題がある。

これらの問題を解決するために、Law-of-the-Jungle (LOJ) アルゴリズムが提案されている [14]。LOJは従来の競合学習方式より少ないコードベクトル更新により最適なコードブックが得られることが示されている。しかしながら、大きなデータサイズの問題を単一のプロセッサにより処理した場合、LOJアルゴリズムをもってしても依然として長い処理時間が問題となっている。このため、LOJによるコードブック生成のさらなる速度向上が求められている。

本論文では、最適コードブックの高速生成を目的として、並列LOJアルゴリズムを提案する。提案アルゴリズムは、連続更新手法を導入することによりメモリ分散型並列計算機において高い並列処理効率を実現する。

本論文の構成は以下の通りである。2節ではベクトル量子化とLOJアルゴリズムを用いたコードブック生成に関して述べる。3節では、並列LOJアルゴリズムとその並列性に関して議論する。4節では、汎用並列計算機IBM SP2、NEC TX-7/AzusA、及びPCクラスタを用いて実験した結果を基に性能評価を行う。5節では、結論と今後の課題を述べる。

2 LOJアルゴリズム

2.1 VQのためのコードブック生成

ベクトル量子化では、k次元のユークリッド空間$\boldsymbol{R}^k$中のベクトルを有限個のコードベクトルによって近似する。コードベクトルの集合をコードブックと呼ぶ。ベクトル量子化器はコードブック$\boldsymbol{Y} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_N\}$とそれぞれのコードベクトル$\boldsymbol{y}_i$に対応した領域$S_i$の集合$\boldsymbol{S} = \{S_1, S_2, ..., S_N\}$により定義される。ベクトル量子化$Q(\boldsymbol{x})$では、次式に示すように, 領域$S_i$に含まれる入力ベクトル$\boldsymbol{x}$が、コードベクトル$\boldsymbol{y}_i$によって近似される。

$$Q(\boldsymbol{x}) = \boldsymbol{y}_i \quad \text{if} \quad \boldsymbol{x} \in S_i \quad \text{for} \quad i = 1, 2, ..., N \tag{1}$$

ここで$\boldsymbol{x} = \{x_1, x_2, ..., x_k\} \in \boldsymbol{R}^k$は$k$次元の入力ベクトルである。$\boldsymbol{x}$を$\boldsymbol{y}_i$により置き換えたことによる歪み$d(\boldsymbol{x}, \boldsymbol{y}_i)$は、ユークリッド距離として次式により与えられる。

$$d(\boldsymbol{x}, \boldsymbol{y}_i) = \|\boldsymbol{x} - \boldsymbol{y}_i\|. \tag{2}$$

$p(\boldsymbol{x})$を$\boldsymbol{x}$の確率密度関数とすると、平均2乗誤差 (MSE) は次式により求められる。

$$MSE(Q) = \sum_{i=1}^{N} \int_{S_i} d^2(\boldsymbol{x}, \boldsymbol{y}_i) p(\boldsymbol{x}) d\boldsymbol{x}. \tag{3}$$

最適なベクトル量子化を行うためには、MSEが最小となるように領域$\boldsymbol{S}$とコードブック$\boldsymbol{Y}$を決定する必要があるが、殆どのアプリケーションでは入力の確率密度関数が不明である。このため、与えられた入力ベクトルに基づいた学習により確率密度関数を推定し、MSEが最小となるコードベクトル配置を決定する必要がある。

2.2 LOJアルゴリズム

Law-of-the-Jungle(LOJ) アルゴリズム [14] は、代表的な競合学習アルゴリズムであるコホネン学習 [12] を拡張したものである。コホネン学習では、入力ベクトルに対して最近傍のコードベクトルが探索され、競合の勝者である最近傍コードベクトルが入力ベクトル方向へ移動される。この探索、及び位置更新からなる手順を学習と呼ぶ。学習ステップを繰り返すことにより、コードベクトルの配置は入力ベクトルの分布を反映したものとなる。しかし、コードベクトルの初期配置によっては勝率に偏りが生じ、勝率の低いコードベクトルが量子化誤差減少に十分に貢献しない場合がある。また勝率の低いコードベクトル近傍には入力が少ないため、このようなコードベクトルが適切な位置へ移動するには、多くの学習ステップが必用となる。さらに、勝率が零のコードベクトルは量子化に全く使われないばかりか、入力により更新される可能性も極めて小さい。

LOJアルゴリズムは以上の問題を解決するために、コードベクトル間の勝率を均一化する。LOJアルゴリズムでは、勝率の低いコードベクトルを取り除き、勝率の高いコードベクトルを2分割することにより勝率の均一化を実現する。これは、入力が少ない領域のコードベクトルを入力の多い領域へのジャンプさせることと等価である。ジャンプは以下の条件を満たす場合に起こる。

(勝者が最大勝率で、かつそれが閾値を越えている)、または
(最小勝率が閾値を下回り、かつ勝者が最大勝率である)

ここで、閾値はユーザにより与えられる。

図1はLOJアルゴリズムの概要である。LOJアルゴリズムは、最近傍コードベクトル探索、コードベクトル更新、MSE計算の3ステージからなる。最近傍コードベクトル探索ステージでは、入力ベクトルとの距離計算により入力に対する勝者となるコードベクトルを探索する。次のコードベクトル更新ス

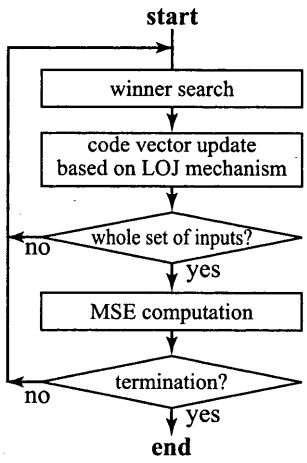

図 1: LOJ アルゴリズムの概要

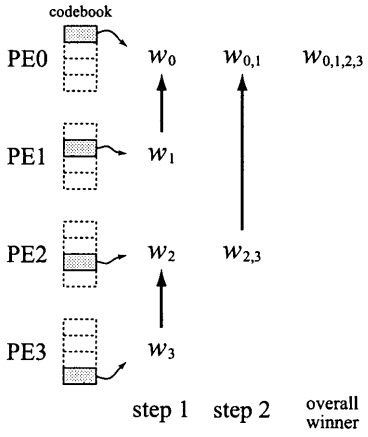

図 2: コードベクトル並列による最近傍ベクトル探索

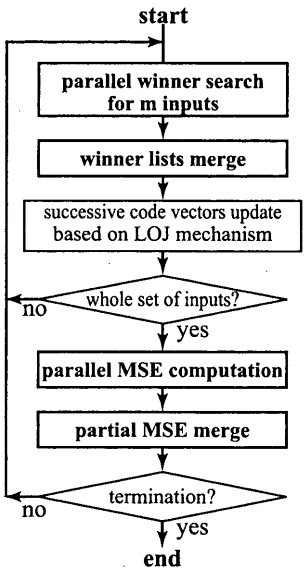

図 3: 入力ベクトル並列アルゴリズムの概要

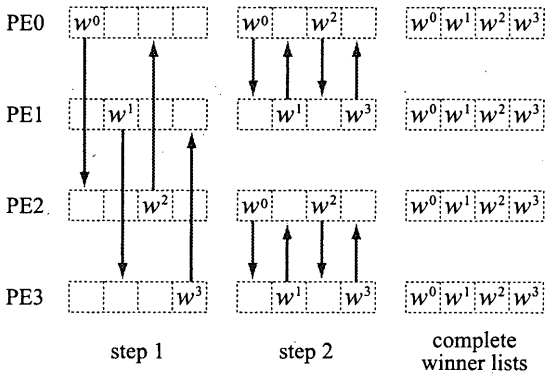

図 4: 最近傍コードベクトル集約処理

テージでは、勝者を更新する。ジャンプ条件を満たした場合、ジャンプ処理が行われる。1 入力ベクトルセットの入力完了後に、MSE 計算ステージにおいて MSE 計算を行う。最後に、MSE の収束評価や、入力ベクトルセットの入力回数条件等により処理終了を判定する。終了条件を満たさない場合は、最近傍コードベクトル探索ステージに戻る。

本論文では最近傍コードベクトル探索ステージと、MSE 計算ステージが高い並列性を有していることに着目する。次節では、これらを並列化した並列 LOJ アルゴリズムを提案する。

3 並列 LOJ アルゴリズム

3.1 LOJ アルゴリズムにおける並列性

LOJ アルゴリズムにおける各処理ステージの占める割合を評価するため、予備実験を行なった。実験では LOJ アルゴリズムによる画像圧縮とし、256 × 256 画素の LENA 標準画像を用いた。各ベクトルを 4 × 4 画素からなる 16 次元ベクトルとし、コードベクトルは入力ベクトル総数の 25% を用いた。実験に

は450MHzで動作するIntel Celeronプロセッサ、及び384MByteの主記憶からなる計算機を用いた。実験の結果、最近傍コードベクトル探索ステージ、及びMSE計算ステージがそれぞれ49.5%の処理時間を占めていることがわかった。

LOJアルゴリズムにおいては、計算処理の大部分を占める最近傍コードベクトル探索ステージ、及びMSE計算ステージが大きな並列性を持っている。最近傍コードベクトル探索ステージの並列性としては、コードベクトル並列性と入力ベクトル並列性が利用可能である。

コードベクトル並列性に基づく最近傍コードベクトル探索では、各計算処理要素(Processing Elements, 以下PEと呼ぶ)が等分割したコードブックの一部分を持ち、入力ベクトル全体を共有するアルゴリズムを考えることができる。図2は4台のPEでの最近傍コードベクトル並列探索の例を示す。この例では、各PEはコードブックの1/4をそれぞれ持つ。同じ入力ベクトルが各PEに与えられ、PEはそれぞれの持つ部分的なコードブック中から局所的な最近傍ベクトルを探索する。この後、図に示すように局所的探索結果が2分木の逆順に集約され、最終的にPE0において最近傍コードベクトルが求められる。

コードベクトル並列性に基づくアルゴリズムの欠点は、局所的最近傍コードベクトルの集約中に多くのPEがアイドル状態になる点である。図2の例におけるステップ2では、PE0のみが局所的探索結果の受信、及び比較を行うのに対し、他のPEはアイドル状態である。並列効率を大きく低下させるこの問題を避けるために、本論文では入力ベクトル並列性に基づきLOJアルゴリズムを並列化する。コードブックを共有する入力ベクトル並列では、並列探索の結果を集約する際に生じる、PE間の負荷のアンバランスの問題が発生しない。

3.2 入力ベクトル並列に基づくLOJの並列化

本節では、図1に示すLOJアルゴリズムを、入力ベクトル並列性に基づき並列化する。並列LOJアルゴリズムの概要を図3に示す。本アルゴリズムでは、最近傍コードベクトル探索ステージ、及びMSE計算ステージが並列化されている。また、新たに勝者リスト集約ステージ、MSE集約ステージを加えた。全入力は複数の部分入力セットに分割され、各部分入力セット毎に並列コードベクトル探索、及び勝者リスト集約を行う。以下の節において、各ステージの詳細を述べる。

3.2.1 最近傍コードベクトル並列探索ステージ

利用可能なPE数をn_pとし、入力ベクトルセットをサイズ$m(m = kn_p$、kは自然数)の部分入力セットに分割する。各PEはコードブック全体を持っており、入力としてm個の入力ベクトルが各PEに共通に与えられる。このm個の入力ベクトルの中から、各PEは$\frac{m}{n_p}$個の異なる入力ベクトルを取り出す。各PEはコードブックと、取り出した$\frac{m}{n_p}$個の入力ベクトルから最近傍コードベクトル探索を並列に行い、$\frac{m}{n_p}$個の最近傍コードベクトルを得る。入力ベクトル個数、コードベクトル個数をそれぞれn_{iv}、n_{cv}とすると、全入力ベクトルセットに対する1PE当たりの計算量は$O(\frac{n_{iv}n_{cv}}{n_p})$となる。

3.2.2 最近傍コードベクトル集約ステージ

PE毎に$\frac{m}{n_p}$個の最近傍コードベクトルが求まった後、これらを各PEに集約する。図4に4PEによる$m = 4$の場合の集約処理を示す。最近傍コードベクトルの集約後、m個の入力ベクトルに対応する最近傍コードベクトルの全てを各PEが持つことになる。対となったPEによる最近傍コードベクトルリストの交換が全て同時に行えるとすると、各ステップにおいて通信される最近傍コードベクトル数によって、全集約時間T_{merge}が決まる。各ステップにおいて通信される最近傍コードベクトル数を全入力セットについて考慮すると、次式を得る。

$$T_{merge} = \frac{n_{iv}}{m} \sum_{k=1}^{\log_2 n_p} \frac{m}{n_p} 2^{k-1} = n_{iv}(1 - \frac{1}{n_p}) < n_{iv} \tag{4}$$

従って、全入力ベクトルセットに対する集約時間は$O(n_{iv})$となる。ここでは1つの通信当たりのセットアップ時間等のオーバーヘッドを無視している。もしセットアップ時間を考慮するのであれば、集約時間はmにも依存することとなる。

3.2.3 コードベクトル連続更新ステージ

このステージは非並列であり、最近傍コードベクトルリストと入力ベクトルに基づいて、各PEが同一の更新処理を行う。コードベクトル更新手法を以下に示す。初めに、全PEがm個の入力ベクトルから同じ入力ベクトルを撰択し、LOJアルゴリズムに基づいて更新処理を行う。最近傍コードベクトルリストに記録されている勝者に対し、ジャンプ処理の後、最近傍コードベクトルはコホネン学習と同様に更新される。この処理を入力の数であるm回分繰り返す。m入力に対し、コードブック不変のまま最近傍コードベクトル探索を行い、その後m個の勝者を連続して更新する。この手法を連続更新と名付ける。一方、1つの入力に対する探索毎に更新を行う従来の手法を単独更新と呼ぶ。入力全体の学習ではn_{iv}回の更新処理を行うため、本ステージの計算量は、$O(n_{iv})$となる。

連続更新により、PEのアイドル状態が避けられる。しかし、m個の入力ベクトルに対し各最近傍コードベクトルを更新無しで連続に探策し、その後対応するコードベクトルを更新する連続更新では、単独更新手法と比べ最近傍コードベクトル探索、及び、更新の順序が異なるため、連続更新アルゴリズムを用いた場合、単独更新のコードベクトル配置と異なる可能性がある。特に、最近傍コードベクトルが同時に複数の入力ベクトルを代表している場合、コードブックの更新結果がよりいっそう大きく異なる場合があると考えられる。その結果、連続更新アルゴリズムを用いた場合、単独更新と比較してMSE収束値が増加することが考えられる。

3.2.4 並列MSE計算・集約ステージ

並列MSE計算ステージでは、各PEが$\frac{n_{iv}}{n_p}$個の入力ベクトルに対し、対応するコードベクトルとの歪みを並列に計算する。このステージの計算量は$O(\frac{n_{iv}n_{cv}}{n_p})$であり、$n_p$に比例した速度向上が得られる。MSE集約ステージでは、1つの代表PEが各PEの計算結果を集約しMSEを求める。計算時間は$O(n_p)$となる。

3.3 並列性の見積り

並列最近傍コードベクトル探索ステージと並列MSEステージの計算量は$O(\frac{n_{iv}n_{cv}}{n_p})$であり、本論文で提案するアルゴリズムにおいて完全に並列処理可能な部分である。一方、残りのステージは非並列部分であり、全体の並列処理効率低下の要因となる。特にMSE集約ステージの計算量$O(n_p)$はPE数の増加に伴って並列処理のオーバヘッドを増大させる。しかし、通信データサイズが小さいために実行時間が非常に短く、その影響は小さいと予想される。また、一般に最近傍コードベクトルリスト集約ステージと連続更新ステージの処理時間は並列処理部と比較して短いため、並列処理効率に与える影響は小さいと考えられる。これらステージの計算量は$O(n_{iv})$であり、並列処理部分の計算量が$n_{iv}n_{cv}$に比例して増大することを考えると、n_{cv}が大きい大規模VQにおいて、これらの非並列処理が並列処理効率の著しい低下をもたらすことは無いと予想される。

4 性能評価

提案するアルゴリズムを評価するために実験を行った。本節では実験結果の詳細と考察を述べる。

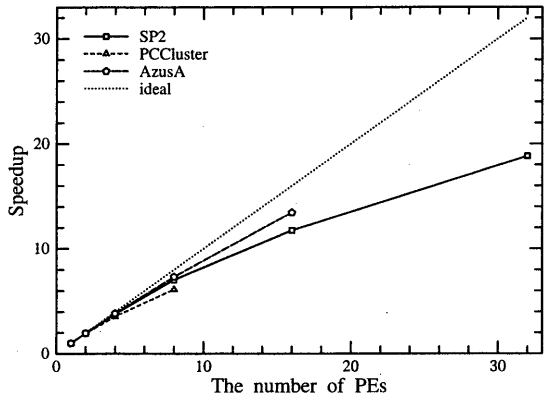

図 5: 並列計算機毎の LENA256 における速度向上率

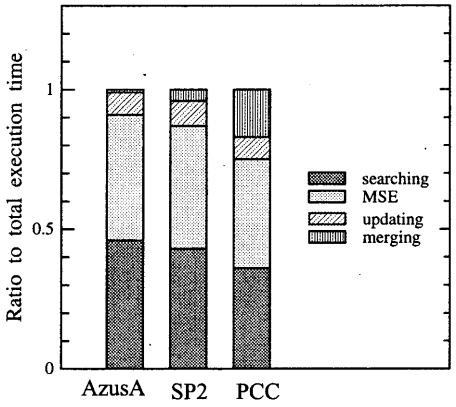

図 6: 並列計算機毎の各ステージの処理時間割合 (8PE)

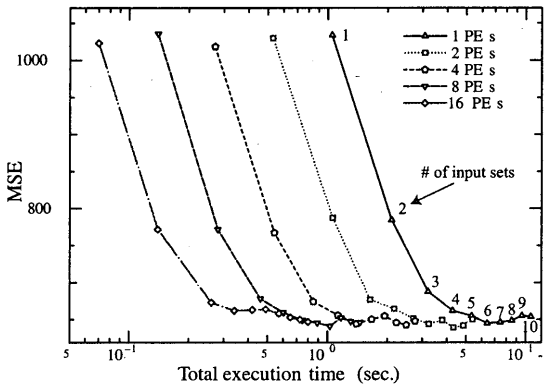

図 7: AzusA における実行時間に対する MSE の収束

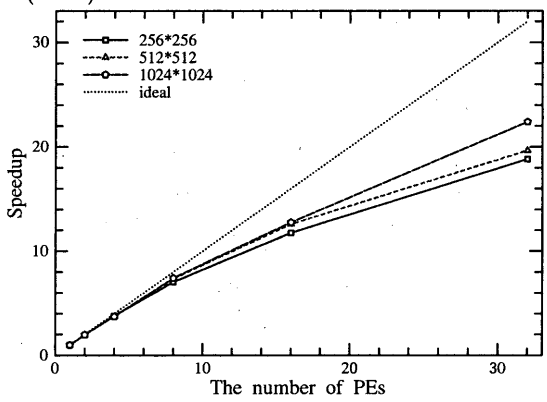

図 8: SP2 における並列 LOJ アルゴリズムの速度向上率

4.1 実験条件

IBM SP2[15](32PE)、NEC TX-7/AzusA(16PE)、及び PC クラスタ (8PE) において、提案する並列アルゴリズムを実装した。SP2 は 66MHz で動作する POWER2 プロセッサ [16] により構成される分散メモリ型並列計算機であり、各ノードはハイパフォーマンススイッチ (HPS) により接続されている。HPS は双方向の多段式結合網で、最大で片方向 40MB/s のバンド幅を持つ。AzusA は 800MHz で動作する Itanium プロセッサにより構成される分散共有メモリ型並列計算機であり、4 つの CPU がメモリを共有し 1 つのセルを構成している。セル間のデータクロスバは最大で片方向 4.2GB/s のバンド幅を持つ。PC クラスタは 1.8GHz で動作する Pentium4 プロセッサにより構成される分散メモリ型並列計算機であり、各ノードは 100Base-TX の SW ハブにより接続され、最大で片方向 12.5MB/s のバンド幅を持つ。AzusA、SP2、PC クラスタの順で、PE 演算性能に対して PE 間通信性能が高いと言える。PE 間通信の実装には MPI メッセージパッシングライブラリを用いた。

実験では 256 階調グレースケール、256 × 256、512 × 512、及び 1024 × 1024 画素の “LENA” 標準画像 (それぞれ LENA256、LENA512、LENA1024 と呼ぶ) を用い、これらを 4 × 4 画素毎のブロックに分割し、各々を 16 次元の入力ベクトルとした。コードベクトル数は入力ベクトル数の 25% とし、連続入力数はコードベクトル数の 25% とした。それぞれ LENA256、LENA512、及び LENA1024 より生成した入力ベクトルセットを MSE が十分に収束する回数である 50 回入力し、1 入力ベクトルセット毎に MSE 計算を行った。PE 数を 2^n に従い変化させ並列 LOJ アルゴリズムによるコードブック生成を行なった。

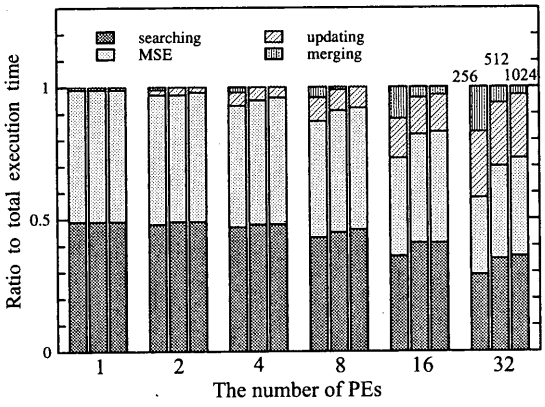

図 9: SP2 における各ステージの処理時間の割合

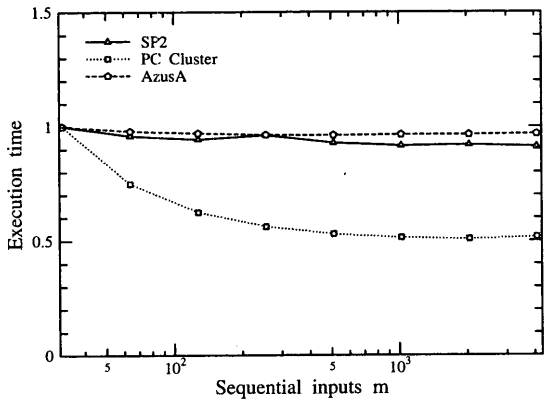

図 10: 連続入力数の違いによる実行時間の変化

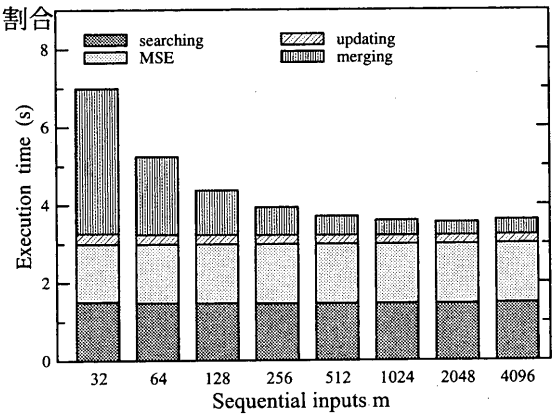

図 11: 連続入力数による実行時間の内訳 (PC クラスタ)

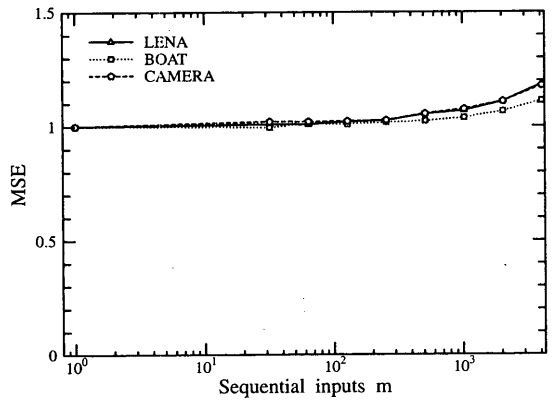

図 12: 連続入力数による MSE 収束値変化

4.2 結果と考察

図 5 に、並列計算機毎の並列 LOJ アルゴリズムの速度向上を示す。図中の点線は理想的な速度向上率を示している。いずれの並列計算機においても、高い並列処理効率が得られた。これは、並列 LOJ アルゴリズムが並列処理部分に比べて非並列処理部分、通信処理部分の割合が非常に小さく、全体の処理効率に与える影響が少ないためと考えられる。しかし、並列計算機によって得られる速度向上率が異なるのは、通信処理部分の占める割合が異なるためと考えられる。PE 数が同一の場合、AzusA、SP2、PC クラスタの順で高い速度向上が得られた。これは、PE 間通信性能の高い並列計算機ほど高い速度向上率が得られることを示している。

図 6 に、各並列計算機の 8PE を用いた場合の LENA256 に対する処理時間の内訳を示す。横軸には画像サイズを示し、実行時間内訳は総実行時間を 1 とした場合の各処理ステージの占める割合を示している。図 6 からわかるように、並列計算機の通信能力の違いにより、全計算時間に占める通信時間の割合が異なる。これは、PE 演算性能/PE 間通信性能のバランスの違いによるものである。このことから、PE 演算性能に対して、PE 間通信性能が高い並列計算機程、台数効果の無い通信処理部の占める割合が低下し、高い速度向上率が得られている。

図 7 は、LENA256 において AzusA の 1PE から 16PE による並列実行での MSE 収束過程を表したものである。横軸は計算時間を示す。PE 台数に反比例して実行時間が減少すると共に、ほぼ同様の MSE 収束値が得られた。これは、連続更新手法を導入した場合でも MSE 収束値の大きな違いが無いことを示している。LENA512、及び LENA1024 についても同様の結果が得られた。また、SP2、PC クラスタにおいても同様の結果が得られた。

図 8 に、SP2 において、画像サイズを変えた場合の並列 LOJ アルゴリズムの速度向上を示す。図中の

図 13: 遂次 LOJ アルゴリズムによる復元画像

図 14: MSE 増加が最大の復元画像 (連続入力数 4096)

点線は理想的な速度向上を示している。本論文で提案したアルゴリズムは、32PE の場合に LENA256、LENA512、LENA1024 に対して、それぞれ 18.8 倍、19.6 倍、22.4 倍の速度向上を達成した。並列処理効率はそれぞれ 0.59、0.61、0.70 であった。この高い並列処理効率は、並列処理部分に比べて非並列処理部分の割合が非常に小さく、全体の処理効率に与える影響が少ないためと考えられる。AzusA、PC クラスタにおいても同様の結果が得られた。

図 9 に、LENA256、LENA512、及び LENA1024 に対するコードブック生成における、各ステージでの処理時間の内訳を示す。1PE 当たりの計算量が最も少ない 32PE 実行時においても、最近傍コードベクトル探索ステージ、及び並列 MSE 計算ステージの計算時間が処理全体に対して大きな割合を占めている。また、3.3 節の議論に基づき、コードベクトル数と連続入力数の比を一定にしたままで入力ベクトル数を 2 倍にすると、並列化部分の計算量増加が非並列部分の増加の 4 倍となり、さらに並列処理効率が向上すると予想される。大規模な画像を処理した場合の方が、得られる速度向上が大きいのはこのためである。以上から、本論文で提案するアルゴリズムは、入力ベクトル数の多い規模の大きなベクトル量子化に向いていると言える。AzusA、及び PC クラスタにおいても同様の結果が得られた。

図 10 に SP2、AzusA、及び PC クラスタの 8PE において、連続入力数の変化による実行時間の推移を示す。連続入力数が 32 の場合の実行時間を 1 とした。横軸は連続入力数、縦軸は実行時間を示す。連続入力数の増加と共に実行時間が減少する結果が得られた。これは、処理全体において行なわれる通信回数が連続入力数に反比例するため、連続入力数が大きい程通信の発生回数が減り、通信オーバーヘッドが減少したことで全体の実行時間が減少したものであると考えられる。特に実行時間の減少が大きい PC クラスタの実行時間内訳を図 11 に示す。横軸は連続入力数、縦軸は実行時間を示している。最近傍コードベクトル探索ステージ、並列 MSE 計算ステージ、及びコードベクトル連続更新ステージは連続入力数によらず一定である。一方で、通信処理時間が連続入力数の増加と共に減少する。実行時間のみが減少する結果は SP2、AzusA においても同様であった。これは連続入力数増加によって、通信回数が減少し通信オーバーヘッドが減少したためと言える。

図 12 に連続入力数による MSE 収束値の変化を示す。実験には LENA256、BOAT、及び CAMERA 標準画像を用いた。横軸は連続入力数、縦軸は MSE 収束値を示す。連続入力数の増加に伴い MSE 収束値が増加する。連続入力数が最大の 4096 において、MSE は約 10〜20%の増加となる。これは連続入力によってコードベクトルの更新誤差が生じるためである。

次に、連続入力手法を用いない場合の復元画像を図 13 に、連続入力数を最大の 4096 とし、MSE が最も増加した場合の復元画像を図 14 に示す。2 つの画像に大きな違いは見られず、画素値差分を比較した場合でも差はほとんど見られなかった。このことから、本論文で提案するアルゴリズムは、最適な連続入力数を用いることにより、MSE 増加を抑え、画質の劣化をほとんど無視できる状態で高速に最適コードブックを生成することが可能であると言える。

5 まとめ

本論文では、メッセージパッシング機構を有する分散メモリ型並列計算機のための並列LOJアルゴリズムを提案した。本アルゴリズムは、従来のコードベクトル更新手法を拡張した連続更新手法を導入することにより、MSE収束性能を劣化させること無しに、高い入力ベクトル並列性を利用している。IBM SP2、NEC AzusA、及びPCクラスタを用いた実験により、画像圧縮のための中規模なベクトル量子化において優れた並列処理効率が得られた。このことは本論文で提案するアルゴリズムが高い並列性を持つことを示している。また、最適な連続入力数を用いることにより、MSE増加を最小に抑え、連続入力手法を用いない場合とほぼ同等の画像を得ることが可能である。これにより、本論文で提案したアルゴリズムは高速に最適コードブックを得ることが可能な並列アルゴリズムであると言える。

参考文献

[1] B.Ramamurthi and A.Gersho. Classified vector quantization of images. *IEEE Transactions on Communications*, COM-34(11):1105–1115, November 1986.

[2] N.Nasrabadi and Y.Feng. Vector quantization of images based upon the kohonen self-organizing feature maps. *Proceedings of the IEEE International Conference on Neural Networks*, 1:101–108, 1988.

[3] P.C.Cosman, K.L.Oehler, E.A.Riskin, and R.M.Gray. Using vector quantization for image processing. *Proceedings of the IEEE*, 81(9):1326–41, September 1993.

[4] O.T.C.Chen, B.J.Sheu, and W.C.Fang. Image compression using self-organizing networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 4(5):480–489, October 1994.

[5] C.Amerijckx, M.Verleysen, P.Thissen, and J.D.Legat. Image compression by self-organized kohonen map. *IEEE Transactions on Neural Networks*, 9(3):503–507, May 1998.

[6] Paul Ning and Lambertus Hesselink. Vector quantization for volume rendering. *Proceedings of 1992 Workshop on Volume Visualization*, pages 69–74, October 1992.

[7] Boon-Lock Yeo and Bede Liu. Volume rendering of dct-based compressed 3d scalar data. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):29–43, March 1995.

[8] A.Gersho and R.M.Gray. *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, 1992.

[9] Y. Linde, A. Buzo, and R. M. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, (1):84–95, January 1980.

[10] S.P.Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, IT-28(2):129–137, March 1982.

[11] S.Grosberg. Adaptive pattern classification and universal recoding:i.parallel development and coding of neural feature detectors. *Biological Cybernetics 23*, pages 121–134, March 1976.

[12] T. Kohonen. *Self-Organization and Associative Memory.* Springer-Verlag, Berlin Heidelberg, 1989.

[13] H. Nielsen. *Neurocomputing.* Addison-Wesley, Redwood City, 1990.

[14] Taira Nakajima, Hiroyuki Takizawa, Hiroaki Kobayashi, and Tadao Nakamura. Kohonen learning with a mechanism,the law of the jungle,capable of dealing with nonstationary probability distribution functions. *IEICE Transactions on Information and Systems*, E81-D(6):584–591, 1998.

[15] T. Agerwala, J. Martin, J. Mirza, D. Sadler, D. Dias, and M. Snir. SP2 system architecture. *IBM System Journal*, 34(2):152–184, 1995.

[16] S. W. White and S. Dhawan. POWER2: next generation of the risc system/6000 family. *IBM System Journal*, 38(5):493–502, 1994.