ハイパフォーマンス
コンピューティング 91－12
（2002．8．21）

ベクトル量子化のためのコードブック生成並列処理に関する研究

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 ‡新潟大学総合情報処理センター
§東北大学情報シナジーセンター

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。32個の計算ノードを用いて並列コードブック生成実験を行った結果、27.4倍の高いスケーラビリティが得られた。

Parallel Codebook Generation for Optimal Vector Quantizer

Shintaro Momose†, Kentaro Sano†, Hiroyuki Takizawa‡, Taira Nakajima†,
Clecio Donizete Lima†, Hiroaki Kobayashi§, and Tadao Nakamura†

†Graduate School of Information Sciences, Tohoku University
‡Integrated Information Processing Center, Nigata University
§Information Synergy Center, Tohoku University

Vector quantization is an attractive technique for lossy data compression, which has been a key technology for data storage and/or transfer. So far, various algorithms have been proposed to design optimal codebooks presenting quantization with minimized errors. In particular, the Law-of-the-Jungle(LOJ) learning algorithm has been proposed to achieve rapid codebook design by algorithmic improvements. However, its acceleration is still required when large data sets are processed on a single computer. Therefore, a scalable parallel codebook design algorithm for parallel computers is required. This paper presents a parallel algorithm for the LOJ learning, suitable for distributed-memory parallel computers with a message-passing mechanism. Experimental results indicate a high scalability of the proposed parallel algorithm on the IBM SP2 parallel computer with 32 processing elements.

1 はじめに

近年、画像、動画、ボリュームデータ等の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、スカラー量子化(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で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CPU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。

並列計算機による並列処理は、計算時間の短縮に有効な手法である。Keshabらは、自己組織化マップ[2][12]におけるニューラル学習に基づいたLBGアルゴリズム[9]を並列化した[15]。Keshabらのアルゴリズムは、コホネン学習よりも少ないコードベクトル更新により、同程度の歪みとなるコードブックを生成可能である。し

かし、必要なコードベクトル更新が更に少なくて済むLOJアルゴリズムの方が計算時間の点で優れている。またKeshabらの研究では並列処理の速度向上に関して詳細な評価がなされていない。

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の32ノード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アルゴリズムによるコードブック生成処理

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。

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

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$\boldsymbol{y}_i$で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。

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

$p(\boldsymbol{x})$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。

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

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$\boldsymbol{Y}$を決定する必用があるが、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き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、探策、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。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。また、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分割と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の、入力の多い領域へのジャンプとみなすことができる。ジャンプは以下の条件を満たす場合に起こる。

(勝者の勝率P_iが最大勝率で、かつ閾値を越えている)、または

(最小勝率P_iが閾値を下回り、かつ勝者が最大勝率である)

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策、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策ステージでは、入力ベクトルとの距離計算により入力に対する勝者であるコードベクトルを探策する。次のステージでは、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策ステージに戻る。

本論文では最近傍コードベクトル探策ステージと、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アルゴリズムにおいて、計算処理の大部分を占める最近傍コードベクトル探策ステージ、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策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策では、各計算処理要素(Processing Elements, PE)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4PEでの最近傍コードベクトル並列探策の例を示す。この例では、各PEはコードブックの1/4を持つ。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策する。この後、図に示すように局所

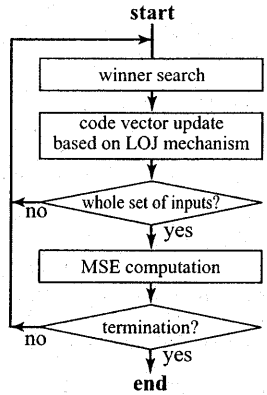

図 1: LOJ アルゴリズムの概要

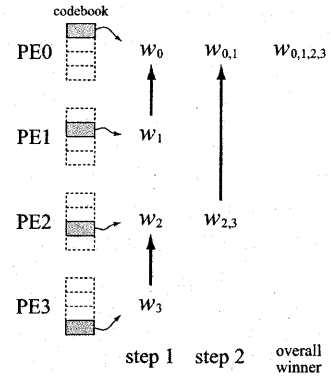

図 2: コードベクトル並列による最近傍ベクトル探策

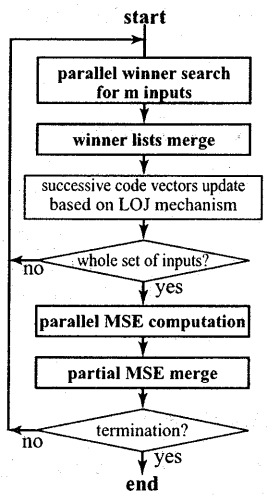

図 3: 入力ベクトル並列アルゴリズムの概要

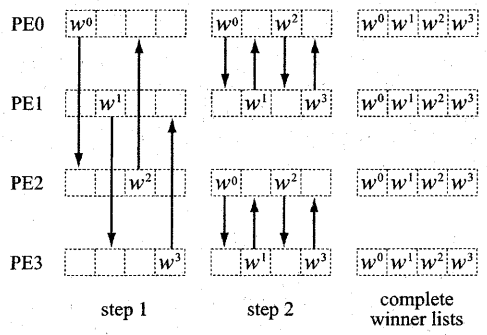

図 4: 最近傍コードベクトル集約処理

的探策結果が 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 のみが局所的探策結果の受信、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、局所的探策に起因する集約中のアイドル問題がない。

3.2 入力ベクトル並列による並列 LOJ

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策ステージ、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策、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策ステージ

n_p を PE 数とし、全入力 n_{iv} 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策を

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。

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

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策を行い、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策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策、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異なる場合があると考えられる。

連続更新による量子化誤差への影響を評価するため、コホネン学習の連続更新による予備実験を行った。実験では、図 5 に示す 256 × 256 画素の “LENNA” 標準画像を 4 × 4 画素ブロックに分割し、これらを 4096 個の 16 次元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%(1024 個) とした。入力ベクトルセットを 50 回入力し、MSE を求めた。

図 6 に、連続入力数 m に対する MSE の収束値を示す。実線は MSE 収束値を示し、左側の y 軸が対応する。破線は複数の入力ベクトルを代表しているコードベクトルの割合を示し、右側の y 軸が対応する。

図 6 に示されるように、m が 256 以上 (コードベクトル数の約 25% 以上) では、m の増加に伴い MSE が増加している。特に、複数の入力ベクトルを代表しているコードベクトルの割合が 0.5 を超えると MSE が増大する結果が得られている。これら MSE 収束値とコードベクトルの複数代表の割合に対する相関を考慮すると、MSE の増大は同一のコードベクトルが多数の入力ベクトルの勝者となる場合の更新処理の不正確さによるものと考えられる。

しかし、m がコードベクトルの 25% 以内であれば、単独更新と同様の MSE 収束値が得られており、小さな m は MSE 収束性に与える影響が小さいと考えられる。本論文では、このような m を並列アルゴリズムに用いる。

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策ステージと並列 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: LENNA 標準画像

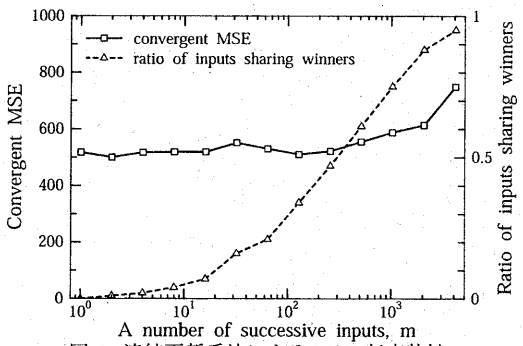

図 6: 連続更新手法による MSE 収束特性

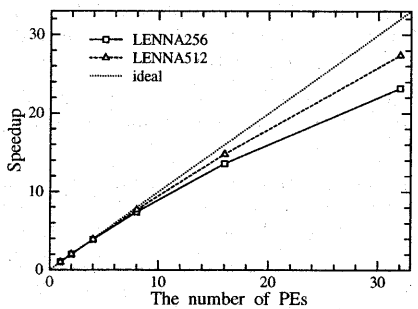

図 7: 並列 LOJ アルゴリズムによる速度向上率

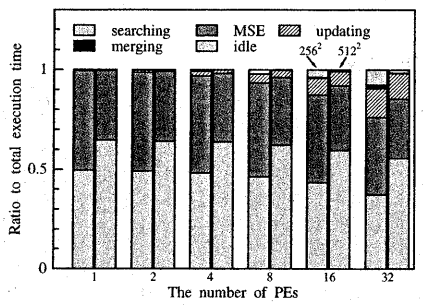

図 8: 各ステージの処理時間の割合

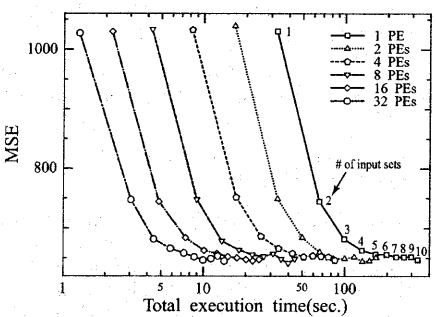

図 9: 実行時間に対する MSE の収束

4.1 実験条件

32 台の PE を有する IBM SP2[16] 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 [17] 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。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 又は 512 × 512 画素の “LENNA” 標準画像 (それぞれ LENNA256、LENNA512 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セットとした。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 m は 3.2.3 節の考察に基づきコードベクトル数の 25% とした。従って、LENNA256 の場合は n_{cv}=1024、m=256 とし、LENNA512 では n_{cv}=4096、m=1024 とした。それぞれ LENNA256、LENNA512 より生成した入力ベクトルセットを 10 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、1、2、4、8、16、32 と変化させて並列 LOJ によるコードブック生成時間を測定した。

4.2 結果と考察

図 7 に並列 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を示す。本論文で提案したアルゴリズムは、32PE の場合に LENNA256、LENNA512 に対して、それぞれ 23.2 倍、27.4 倍の速度向上を達成した。並列処理効率はそれぞれ 0.72、0.86 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。図 8 に、LENNA256、LENNA512 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。図中のアイドル時間は PE 間の同期を取るための待ち時間である。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策ステージ、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。LENNA256よりもLENNA512に対す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。

図9はLENNA256において1PEから32PE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収束性が得られた。これは、連続更新手法によるMSE収束性能の大きな低下が無いことを示している。LENNA512についても同様の結果が得られた。

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。32PEからなるIBM SP2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。512 × 512画素の画像を用いた実験では、32PEでの速度向上は27.4倍であり、この時の並列処理効率は0.86であった。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。

今後の課題は次の通りである。1) PCクラスタを含めた他の並列計算機上での様々なデータセットによる実験、2) 非並列であるコードベクトル更新ステージの並列化、3) ボリュームデータ圧縮等の大規模ベクトル量子化への本アルゴリズムの適用。

謝辞

本研究の一部は日本学術振興会科学研究費補助金(若手研究(B)課題番号13780183)の補助を受けた。

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] Keshab K. Parhi, Frank H. Wu, and Kalyan Genesan. Sequential and parallel neural network vector quantizers. *IEEE Transactions on Computers*, 43(1):104–109, January 1994.

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

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