I-034

圧縮データ上での画像処理アルゴリズムの開発
Development of image processing algorithm on compressed data

赤松 孝則† 下馬場 朋禄† 角江 崇† 伊藤 智義†
Takanori Akamatsu Tomoyoshi Shimobaba Takashi Kakue Tomoyoshi Ito

1. 研究背景と目的

近年，画像や映像の高精細化が進み，画素数の多い画像に対して様々な画像処理が行われている．しかし，画像処理の計算量は画像の画素数に比例するため，高精細な画像では膨大な計算コストとなってしまう．そこで，何らかの方法で画素数を減らすことができれば，処理の高速化が期待できる．画素数を減らす方法の一つに，画像のデータをより小さなデータ量に変換する圧縮処理がある．画像の圧縮処理には JPEG や PNG など様々な方式があり，ある程度の品質を保ちながら大幅にデータ量を削減することができる．しかし，圧縮処理を行った画像は形式が変化するため，画像処理を行うためには，圧縮前の形式に復元する展開処理が必要となる．したがって，画像が圧縮された状態で実行可能な画像処理アルゴリズムを構築することで処理の高速化が期待できる．

そこで本研究では，圧縮処理をランレングス圧縮，画像処理をフーリエ変換として，圧縮データに対する画像処理アルゴリズムを構築した．さらに，非圧縮画像に対するフーリエ変換と，圧縮処理と圧縮画像に対するフーリエ変換を合わせた処理との計算時間の比較を行った．本稿では，ランレングス圧縮したデータに対するフーリエ変換をランレングスフーリエ変換と呼び，ランレングスフーリエ変換を用いた 2 次元フーリエ変換を提案する．

2. ランレングスフーリエ変換の定式化

ランレングス圧縮とは，図 1 に示すようにデータを数値とその連続回数の組で表現する圧縮方法であり，圧縮データから元データを完全に復元することができる可逆圧縮である[1].

図 1　ランレングス圧縮

本研究では，ランレングスフーリエ変換の定式化のため式(1)に示すようにランレングス圧縮されたデータの展開の定式化を行った．

$$a_k = b_{2j}$$
$$j = \min\left\{ l \,\middle|\, k < \sum_{m=0}^{l} b_{2m+1} \right\} \tag{1}$$

式(1)において a は展開したデータ，b は圧縮データ，j は圧縮データにおける連続回数を足し合わせて展開したいデータの位置を何組目で越えるかということを表している．

式(2)に定式化したランレングスフーリエ変換を示す．

$$\mathrm{Re}\left[\hat{a}_t\right] = \sum_{p=0}^{\frac{M}{2}-1} b_{2p} \frac{\sin\left(\frac{b_{2p+1}+1}{N}t\pi\right)}{\sin\left(\frac{t\pi}{N}\right)} \cos\left[\frac{t\pi}{N}\left(2\sum_{m=0}^{p-1} b_{2m+1} + b_{2p+1}\right)\right]$$
$$\mathrm{Im}\left[\hat{a}_t\right] = -\sum_{p=0}^{\frac{M}{2}-1} b_{2p} \frac{\sin\left(\frac{b_{2p+1}+1}{N}t\pi\right)}{\sin\left(\frac{t\pi}{N}\right)} \sin\left[\frac{t\pi}{N}\left(2\sum_{m=0}^{p-1} b_{2m+1} + b_{2p+1}\right)\right] \tag{2}$$

式(2)において $\mathrm{Re}[\hat{a}]$はフーリエ変換後の実部，$\mathrm{Im}[\hat{a}]$はフーリエ変換後の虚部，Mは圧縮データのサイズ，Nは圧縮前のデータサイズを表している．式(2)の右辺において一つ目の積算の回数は圧縮データのサイズに比例することがわかる．高圧縮率のデータであれば M の値が小さくなるため，積算の回数が減少し計算時間を短縮することができる．また，式(2)を用いてフーリエ変換を行った場合，フーリエ変換後の周波数領域のデータは非圧縮データとなる．

3. 画像に対するフーリエ変換

画像は 2 次元のデータであるため周波数領域への変換を行う場合，2 次元フーリエ変換が用いられる．図 2 に一般的な 2 次元フーリエ変換と，提案手法による 2 次元フーリエ変換の処理の流れを示す．

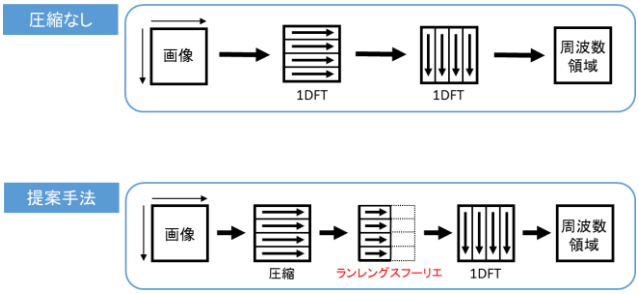

図 2　2 次元フーリエ変換

† 千葉大学大学院工学研究科　Graduated School of Engineering Chiba University

2 次元フーリエ変換とは，図 2 圧縮なしに示すように，フーリエ変換を画像の水平方向に行った後，垂直方向に対して行う処理である．本研究で提案する圧縮画像に対する 2 次元フーリエ変換では，画像の水平方向を 1 つのデータとみなして圧縮処理を行い，ランレングスフーリエ変換を行う．ランレングスフーリエ変換後のデータは非圧縮であるため，そのままの状態で垂直方向に対してフーリエ変換を行うことで 2 次元フーリエ変換を実現することができる

4. 実装結果

723×723[pixel]の画像 76 枚を使用し，CPU および GPU で実装したランレングス圧縮とランレングスフーリエ変換の処理時間を測定した．また，非圧縮画像に対して GPU 用高速フーリエ変換ライブラリである cuFFT[2]を用いた 2 次元フーリエ変換の処理時間を測定した．以降，圧縮率とは圧縮前のデータサイズを圧縮後のデータサイズで除算した値である．

図 3 に CPU および GPU によるランレングス圧縮とランレングスフーリエ変換の合計処理時間を示す．図 3 から GPU を用いて処理を行うことで，CPU の 100 倍以上の処理速度を実現できることがわかる．

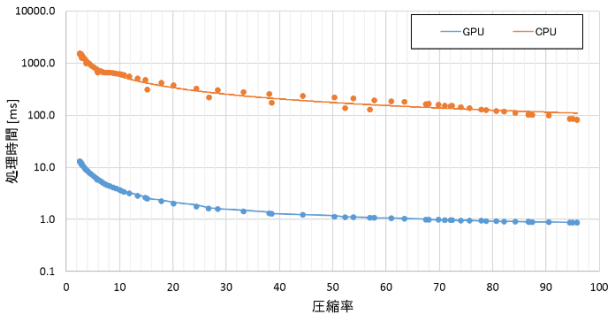

図 3　CPU と GPU の比較

図 4 に GPU を用いたランレングス圧縮とランレングスフーリエ変換それぞれの処理時間を示す．今回測定に使用した画像サイズでは，ランレングス圧縮の処理時間は 0.7[ms]程度となった．また，ランレングスフーリエ変換の処理時間は圧縮率に反比例しており，高圧縮率のデータほど高速な処理を実現いる．今回使用した画像の中で，最も高い圧縮率は 95 であり，この画像に対しては 0.25[ms]でランレングスフーリエ変換を行うことができている．

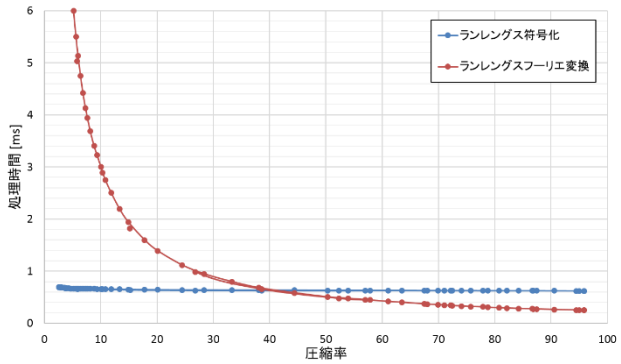

図 4　GPU による処理時間の内訳

図 5 に GPU による提案手法の処理時間と，cuFFT を用いた 2 次元フーリエ変換の処理時間を示す．図 5 より，今回使用した画像サイズでは cuFFT を用いた 2 次元フーリエ変換は 3.5[ms]程度の処理時間となることがわかった．また，圧縮率 10 を超えると提案手法が cuFFT よりも高速に 2 次元フーリエ変換を行うことができ，最も圧縮率の高い画像では，1.6 倍の処理時間を実現することができた．

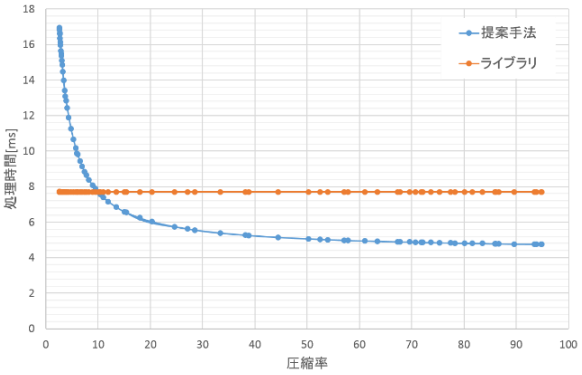

図 5　提案手法とライブラリの比較

5. まとめと今後の展望

ランレングス圧縮したデータの展開およびランレングスフーリエ変換の定式化を行った．CPU および GPU においてランレングスフーリエ変換と提案手法を実装し，GPU を用いて CPU の 100 倍以上の高速化を実現することができた．また，GPU において提案手法による 2 次元フーリエ変換と CUFFT ライブラリによる 2 次元フーリエ変換の処理時間を比較し，最大 1.6 倍の高速化を実現することができた．今後の展望として，より多くの画像処理をランレングス符号化による圧縮データに対して実現することや，異なる圧縮処理による圧縮データに対する画像処理の実装などが挙げられる．

参考文献

[1] Pountain, Dick. "Run-length encoding." Byte 12.6 (1987): 317-319.
[2] ” cuFFT”　https://developer.nvidia.com/cufft (2016 年 5 月現在)