Hilbert-Huang 変換の並列化および GPU による高速化

Pulung Waskito† 三輪 忍† 満倉 靖恵† 中條 拓伯†

1 はじめに

信号解析の分野では，楽曲などの音声信号，株価の変動，平均気温の変動といった，現実の複雑な信号を解析の対象とすることが多い．こうした信号は，時間とともに周波数が変化する，非定常な信号である．非定常信号の解析には，以前は短時間フーリエ変換が用いられることが多かった．しかし，短時間フーリエ変換には時間分解能と周波数分解能との間にトレードオフが存在するため，実際には十分な分解能を得ることができないこともある．そのため，より分解能の高いアルゴリズムがいくつか研究されてきた [1][2].

その 1 つに Hilbert-Huang 変換（HHT）[2] がある．HHT は，比較的新しい解析手法であり，非定常信号に対して優れた分解能を示すことが知られている．その一方で，計算量の多さが問題視されている．HHT のボトルネックとなっている部分は，後述するように，$O(N)$ のアルゴリズムを最内ループとする 2 重ループの構造を持つ．その結果，1 ノードで逐次的に HHT を実行した場合，3 秒程度の音声信号を解析する場合でさえ数時間を要する．

そこで我々は，HHT を並列化し，CUDA[?] で実装する．上述のボトルネック部分を実装した結果，CPU で実行した場合と比べて 27.7 倍の速度向上を得ることができた．本稿ではその詳細について報告する．

本稿の構成を以下に示す．まず次章において，ボトルネック部分を中心に HHT のアルゴリズムを詳しく述べる．続く 3 章では，今回実装した並列 HHT について説明する．評価は 4 章で行い，5 章でまとめる．

2 Hilbert-Huang 変換

HHT は，図 1 に示すように，異なる 2 つのアルゴリズムからなる．1 つは，信号の瞬間的な特性を解析する際に用いられる，ヒルベルト変換（Hilbert Spectral Analysis．以下 HSA とする）である．ただし，HSA には，複数の周波数の信号が混ざっている場合には適用できない，という制約がある．そのため，HSA の前処理として，もう 1 つのアルゴリズム，経験的モード分解（Empirical Mode Decomposition．以下 EMD とする）を入力信号に対して適用するのである．

† 東京農工大学
Tokyo University of Agriculture and Technology

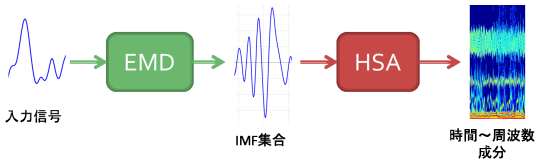

図 1 Hilbert-Huang Transform (HHT) プロセス

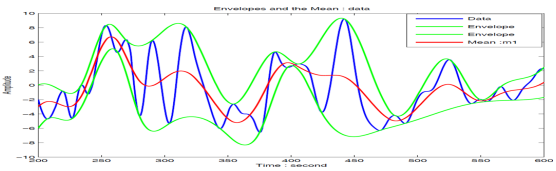

図 2 エンベロープ計算

詳細は省略するが，HSA の処理は EMD のそれよりも圧倒的に負荷が少ない．また，HSA の処理の大部分は，変換を行う信号に対する FFT と，フーリエ空間上での演算結果に対する 逆 FFT とで占められている．FFT，逆 FFT ともに，それらを並列化する研究は長年行われており，CUDA で実装した例も既にある．そこで本稿では，EMD の並列化とその実装のみを議論の対象とする．以下，EMD について詳しく述べる．

2.1 EMD の概要

EMD は，入力信号を固有モード関数 (Intrinsic Mode Function，以下 IMF とする) と呼ばれる信号の集合に分解する手法である．IMF の定義は以下の条件を満たしている信号のことである：

- 極値（極大値・極小値）の数と x 軸との交点の数は一致しているか，または一つ違いである．
- 極大値をつなぐエンベロープ（図 2）と極小値をつなぐそれとの平均は 0 である．

EMD のアルゴリズムを図 3 のフローチャートに示す．まず，入力信号 $\boldsymbol{s_i}$ に対して全ての極大値と極小値を探す（a）．次いで，極大値同士，および，極小値同士ををつなぐエンベロープ (upper_env，lower_env) を 3 次スプライン補間 [3] により求める（b）．結果的に入力信号を囲むような 2 つのエンベロープができる（図 2）．そして，この二つのエンベロープの平均を$\boldsymbol{m_i}$ とし，入力信号と平均との差 $\boldsymbol{h_i} = \boldsymbol{s_i} - \boldsymbol{m_i}$ をとる（c）．

この $\boldsymbol{h_i}$ は protoIMF と呼び，まだ IMF にはなっていないという意味を持つ．最内ループではこの $\boldsymbol{h_i}$ が

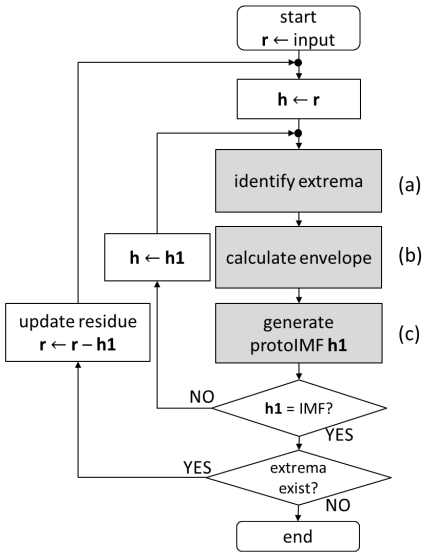

図 3　EMD アルゴリズム

IMF になるまで $\boldsymbol{h_i}$ を入力信号として扱い，上記の処理を繰り返していく．

最初の IMF（IMF1）が求まると，入力信号から $\boldsymbol{r_i} = \boldsymbol{s_i} - \boldsymbol{h_i}$ のように IMF1 との差をとれば，IMF1 信号を除いた信号が得られる．この残差にはまだ複数の周波数の信号が混ざっているため，これを新しい入力信号として上記の処理を繰り返す．残差 $\boldsymbol{r_i}$ に極値がなくなるまで，IMF への分解を続ける．

以下で述べるように，最内ループの各処理の計算量は $O(N)$ である．

2.2　極値決定

入力信号 s_i を $i = 0, \cdots, N-1$ の順に前後の値を比べていくことで極大値・極小値を決定する．順に調べていくため，全体は $O(N)$ の処理になる．

得られた極大値（極小値）の集合を配列に格納する．この配列を使ってエンベロープを計算する．

2.3　エンベロープ計算

エンベロープは 3 次スプライン曲線を用いて作成する．極値 (x_j, y_j) の集合 $j = 1, \cdots, n$ が与えられたとき，それらを結ぶ 3 次スプライン曲線（e_i）は以下の式で計算できる [3].

$$e_i = Ay_j + By_{j+1} + Cy''_j + Dy''_{j+1} \qquad (1)$$

ただし $x_j < i < x_{j+1}$ である．y''_j は各極値 x_j の二階微分の値である．A, B, C, D の変数の説明は省略するが，i と x_j と x_{j+1} の値とで決まる．

実際は各極値の y''_i の値が不明なので，(1) 式を計算する前に，y''_i の値を求めなければならない．y''_i の値は以下の連立方程式を解くことで得られる [3].

$$ay''_{j-1} + by''_j + cy''_{j+} = r_j \qquad (2)$$

同じように説明は省略するが，a, b, c, r の変数はそれぞれ 2 つの極値 x_j と x_{j+1} の値で決まる．(2) 式は以下のような 3 重対角行列で表現できる．

$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} y''_1 \\ y''_2 \\ y''_3 \\ \vdots \\ y''_n \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ \vdots \\ r_n \end{bmatrix} \qquad (3)$$

この行列の解（y''_i）は TDMA（Tri-Diagonal Matrix Algorithm）法 [3] と呼ばれる手法を用いると $O(n)$ で計算できる．TDMA はガウス消去法を簡単化したもので，3 重対角行列に限って，1 回の前進消去と 1 回の後退代入だけで解を求めることができる．

全ての y''_i の値が分かれば，(1) 式によってエンベロープを計算できる．

2.4　protoIMF 生成

入力信号に対して upper_env と lower_env 両方が求まれば，その平均と protoIMF は $O(N)$ で計算できる．そのコードを以下に示す．

```
for( i = 0; i<N; i++ ){
  m[i] = (upper_env[i] + lower_env[i])/2.0;
  h1[i] = h[i]-m[i];
}
```

3　並列 EMD

前章で述べたアルゴリズムを CUDA で実装した．以下その詳細を述べる．

3.1　共有メモリを意識した並列化

CUDA を使った一般的な実装として入力データをブロックごとに分割し，各ブロックが GPU の各 Streaming Multiprocessor(SM) によって並列に実行されるというプログラミングモデルが用いられる．各 SM 内には小量な共有メモリが容易され，同一のブロック内では高速にアクセスすることが出来る．

EMD の並列化において，共有メモリを活用する場合，入力信号を共有メモリに収まる程度のサイズに分割することがまず考えられる．各ブロックは，共有メモリ内のデータに対してのみ，前章で述べた最内ループの処理を行うのが理想である．

しかし，2 章で説明したように，エンベロープ計算では入力信号全ての極値を考慮した上で，それぞれの y''_j を計算する必要がある ((3) 式)．入力信号を単純に分割し，各ブロックが独立にエンベロープ計算すると，分割した領域間のエンベロープが滑らかでなくなってしまう．その結果，IMF に誤差が生じる恐れがある．

そこで，データを分割する際は，ある程度の区間をオーバラップさせる．その動作を図 4 に示す．各ブ

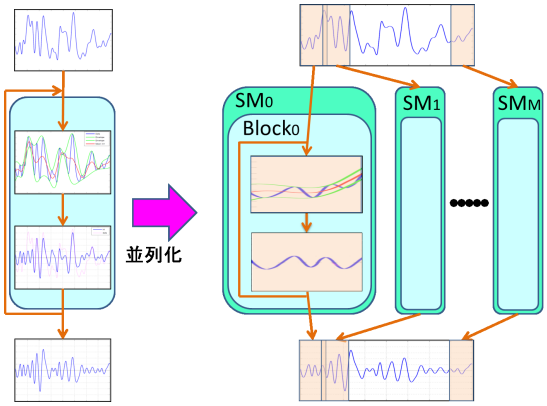

図 4　並列 EMD の概要

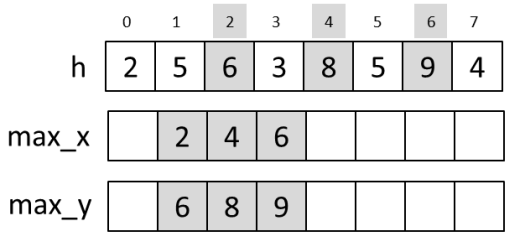

図 5　極値のパディング

ロックは両端のエンベロープをある程度余分に計算しておき，IMF を生成する際は，余分に計算した部分を切り捨てるようにする．

3.2　極値決定

極値決定では，各スレッドが，ある点が極値か否かを判定する．極値か否かはその点の前後の点があれば分かるため，並列に判定することは容易である．

ただし，極値を求めた後の処理には考慮すべき点がある．2.2 で述べたように，次のエンベロープ計算に極値の集合を使用するため，極値だけの配列を生成しておきたい（図 5）．配列 max_x に極値のインデクスを，配列 max_y に極値の値を格納したい．なお，図は極大値の場合であり，灰色の部分が極値を表している．

今回，max_x，max_y のインデクス計算に prefix sum 手法を用いた．図 6 に配列の要素数が 8 の場合の prefix sum の動作を示す．極値の部分が 1，それ以外の部分が 0 となる配列に対して prefix sum を計算する．そうすれば，図のように，各極値が何番目の極値かがわかる．後は，その値をインデクスとして，元の配列（h）から極値だけの配列（max_x，max_y）にデータを移し替えればよい．

prefix sum は $log(n)$ のステップでできるので，極値決定の計算量も $O(log(n))$ になる．

3.3　エンベロープ計算

2.3 で述べたように，エンベロープ計算の際は，3 重対角行列を解く必要がある．3 重対角行列の解法には，

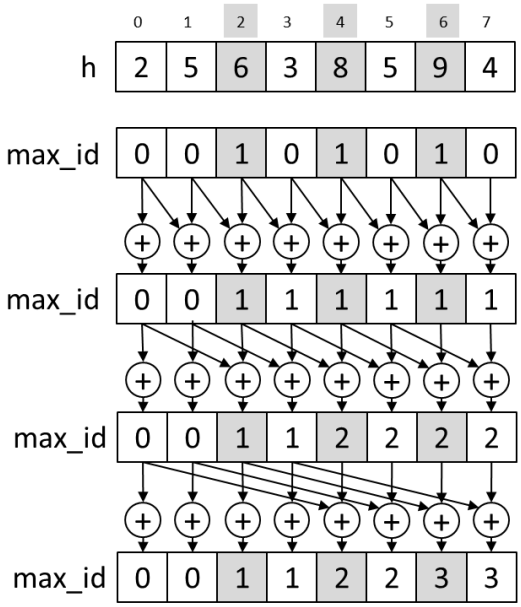

図 6　prefix sum の動作

TDMA ではなく，並列化が効く cyclic reduction [4] を用いる．cyclic reduction では，並列にガウス消去法を行い，各行の項を消していく．項が 1 つになるまで消去を繰り返し，解を求める．

（3）式の 3 重対角行列の各行の項を，その上下の行を用いて消去した結果を 4 に示す．すなわち，2 行目の $a_2y_1'' + b_2y_2'' + c_2y_3'' = r_2$ という式の y_1'' 項を 1 行目の式 $b_1y_1'' + c_1y_2'' = r_1$ によって消去する．同様に，y_3'' 項を 3 行目の式を用いて消去する．この処理は，明らかに各行に対して並列に行える．

$$\begin{bmatrix} b_1' & 0 & c_1' & & 0 \\ 0 & b_2' & 0 & c_2' & \\ a_3' & 0 & b_3' & \ddots & \\ & \ddots & \ddots & \ddots & 0 \\ 0 & & a_n' & 0 & b_n' \end{bmatrix} \begin{bmatrix} y_1'' \\ y_2'' \\ y_3'' \\ \vdots \\ y_n'' \end{bmatrix} = \begin{bmatrix} r_1' \\ r_2' \\ r_3' \\ \vdots \\ r_n' \end{bmatrix} \quad (4)$$

こうして得られた（4）式に対し，上記の処理を繰り返す．これを $log(n)$ 回繰り返すと，（5）式に示す，各行の項が 1 つになるような行列式が出来る．この行列からそれぞれの y_n'' は容易に求めることができる．

したがって，エンベロープ計算も $O(log(n))$ になる．

$$\begin{bmatrix} b_1' & & & & 0 \\ & b_2' & & & \\ & & b_3' & & \\ & & & \ddots & \\ 0 & & & & b_n' \end{bmatrix} \begin{bmatrix} y_1'' \\ y_2'' \\ y_3'' \\ \vdots \\ y_n'' \end{bmatrix} = \begin{bmatrix} r_1' \\ r_2' \\ r_3' \\ \vdots \\ r_n' \end{bmatrix} \quad (5)$$

表 1　評価環境

	CPU	GPU
プロセッサ	Core 2 Duo	TESLA C1060
動作周波数	2.53GHz	602MHz
開発環境	CentOS 5.3	CUDA SDK 2.3
	gcc-4.1.2 -O3	nvcc

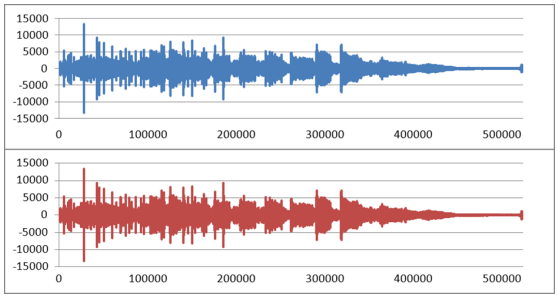

図 7　CPU(上)，GPU(overlap=64)(下)

3.4　protoIMF

protoIMF の生成については， 2.4 のコードにデータ並列性があるので，単純に並列化できる．各スレッドが各点の平均を並列に計算し，それらの protoIMF を並列に生成する．

4　評価結果

今回使用する CPU と GPU の評価環境を表 1 に示す．CPU は Intel Core 2 Duo 2.53GHz を用いる．GPU に関しては，240 個の SP(Stream Processor) からなるシングルカードの TESLA C1060 を使用する．

入力信号として，44.1kHz でサンプリングされた音声信号（Hotel California.wav）を使用する．先頭から 512〜512K 個のサンプリング点を入力とする．オーバラップ区間は 64 点と 128 点の場合を評価した．

1 つのブロックは，512 個のサンプリング点に対して EMD を行う．各ブロックは 256 スレッドを生成して計算する．したがって，各スレッドは，基本的には，2 ポイントのデータを担当することになる．

実際の EMD では IMF の集合を生成するが，簡単のため，今回は IMF1 のみの計算時間を計測した．また，最内ループの終了条件，IMF 判定についても，簡単のため，を行わず，ループが 1,000 回実行されたら IMF になったとみなすものとする．

図 7 にサンプリング点が 512K 個の場合の CPU と GPU(overlap=64) の IMF1 の結果を示す．オーバラップの効果により，両方の差はほとんどない．

図 8 に入力信号のサンプリング点数を変えた場合の CPU と GPU の実行時間を示す．また，速度向上を図 9 に示す．図より，GPU (overlap=64) で実行することで最大 27.7 倍の速度向上が得られた．

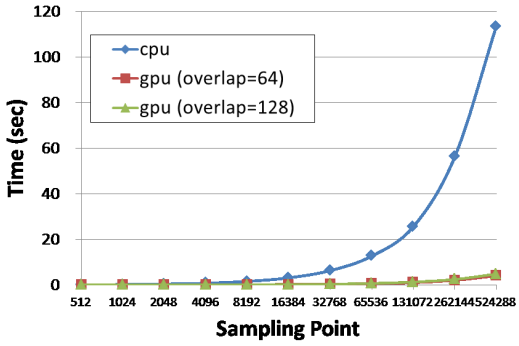

図 8　CPU と GPU の実行時間

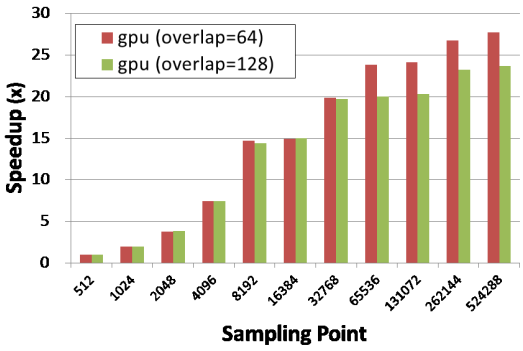

図 9　CPU に対する GPU の速度向上

5　まとめ

今回の EMD の並列化では，最内ループの全ての処理の計算量を $O(log(n))$ に減らすことに成功した．共有メモリを意識した並列化を行うことで，サンプリング点が 512K 個の場合，27.7 倍の速度向上が得られた．また，オーバーラップ区間を設けたことで，並列 EMD の結果は逐次のそれとほとんど同じになることが確認できた．今後は HSA の並列化に取り組む予定である．

参考文献

[1] R.M. Rao and A.S. Bopardikar, ”Wavelet Transforms:Introduction to Theory and Applications”. *Addison Wesley Longman* (1998).

[2] Huang N. E. et al.: ”The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis”. *Proc. R. Soc. Lond. A454, p.903-995* (1998).

[3] McKinley S. and Levine M.: ”Cubic Spline Interpolation”. *College of the Redwoods* (1998).

[4] Bini D.A. and Meini B.: ”The cyclic reduction algorithm: from Poisson equation to stochastic processes and beyond”. *Numer. Algor. vol.31, p.23-60* (2009).