M_030

# ブロック浮動小数点システムのためのデータ配置手法

## A Data Alignment Method for Block Floating Point Systems

浜辺 崇† 坂主 圭史† 武内 良典† 今井 正治†
Takashi Hamabe Keishi Sakanushi Yoshinori Takeuchi Masaharu Imai

## 1. はじめに

ディジタル信号処理システムにおいて，実数データの表現形式は，システムの性能に大きく影響する．実数データの表現形式の 1 つとして，ブロック浮動小数点表現が提案されている．ブロック浮動小数点表現は，処理するデータをいくつかのブロックに分け，ブロックごとに異なる小数点位置を与えることで，固定小数点表現と同等ハードウェア量および演算速度で浮動小数点表現と同等の演算精度を実現している[1]．ブロック浮動小数点表現を使用したシステムにおいて，ブロック内のデータの組合せはデータメモリ上のデータの配置より決定するため，データメモリ上のデータの配置はシステムの演算精度および演算速度に大きく影響する．しかし，ブロック浮動小数点システムのデータ配置に関する研究はまだ報告されていない．ブロック浮動小数点表現システムを設計するためには，設計者がブロック化するデータ配置を決定し，その配置を用いて設計したシステムを解析[3-6]することによって，システムが要求される性能を満たすことを確認する必要がある．しかし，大量の実数データを扱うディジタル信号処理システムにおいては，データ配置の決定とシステムの性能解析の繰り返しには膨大な工数を必要とする．そこで，対象アプリケーションから最適なデータ配置を求める手法が必要とされている．

本稿ではブロック浮動小数点表現を使用したディジタル信号処理システムのデータの配置手法を提案する．提案手法は，Kernighan and Lin アルゴリズム[8]を基に，値の範囲が近いデータをデータメモリ上の近い位置に配置することよって，システムのハードウェア量を最小化する．

以下，第 2 節でブロック浮動小数点について説明し，第 3 節で提案するデータ配置手法を説明する．第 4 節では評価実験について述べ，第 5 節でまとめる．

## 2. ブロック浮動小数点表現

### 2.1 ブロック浮動小数点表現のデータ形式

図 1 にブロック浮動小数点表現のデータ形式を示す．

ブロック浮動小数点表現では，実数値は，仮数を表す $n$ ビットのビット列 $m = (b_{n-1}, b_{n-2}, \ldots, b_0)$ とブロック指数 $p$ で表現される．

実数値 $r$ は式(1)により求められる．

$$r = (-b_{n-1} * 2^{n-1} + \sum_{i=0}^{n-2} (b_i * 2^i)) * 2^p \quad (1)$$

ブロック浮動小数点表現では，同一ブロック内のデータは同一のブロック指数を持つ．したがって，同一ブロック内であれば，固定小数点数として演算することができるため，浮動小数点表現と比較して高速な演算が可能である．

また，ブロックごとに異なる指数を与えることができるため，ブロック指数を適切に与えることによって浮動小数点表現と同等の精度で演算することができる．

### 2.2 ブロック浮動小数点システムのデータ配置問題

ブロック浮動小数点表現では，ブロック指数は設計者が与える必要があるため，ブロックごとに適切なブロック指数を与えなければ，高い精度は確保できない．

また，ブロック浮動小数点表現ではブロック内では小数点位置が固定であるため，絶対値が大きく異なるデータが同一ブロックに含まれている場合，高い精度を確保できない．図 2 はメモリ内部を表し，太線の長方形で囲まれた部分で 1 つのデータの仮数部を表し，斜線部で有効ビットを表すものとする．黒丸は小数点位置とする．図 2 (a) では，絶対値が大きく異なるデータが同一ブロック内にあるため，有効ビットが少なく，高い精度を確保できない．一方，(b) は各データの絶対値の差は(a)より小さいため，(a)と比較して有効ビットが多く，高い精度の演算が可能である．

このように，小数点位置およびブロック内のデータの組合せを決定しなければならないため，ブロック浮動小数点システムの開発は浮動小数点や固定小数点と比較して難しく，適切な設計には膨大な工数を要する．

通常，ブロック浮動小数点システムにおいては，同一ブロック内のデータの仮数部はデータメモリの連続する領域に配置される．これは，仮数とブロック指数の対応を容易にし，小さいサイクル数でブロック間の演算を行うためである．ブロック浮動小数点を使用して設計するシステムにおいては，データメモリ上のデータの配置はブロック化す

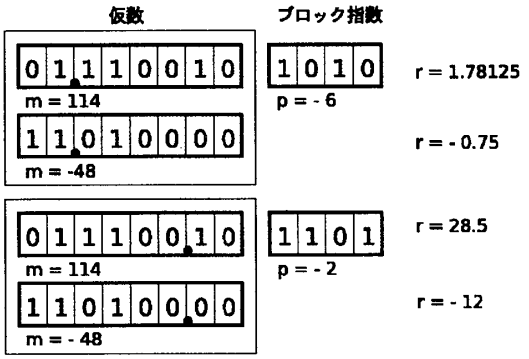


図 1: ブロック浮動小数点の例

† 大阪大学大学院情報科学研究科 情報システム工学専攻

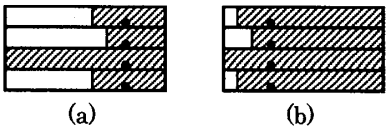


図 2: ブロック浮動小数点のデータ組合せ問題

るデータの組合せを決定するため，システムの演算精度および必要となるメモリサイズに大きく影響する．

また，設計するアプリケーションが配列を含む場合，配列内のデータのインクリメンタルアクセスを可能にするために，配列内のデータは連続する領域に配置する必要がある．

このように，ブロック浮動小数点表現を使用したシステムを設計するときには，データのビット長によるブロック化するデータの組合せと同時に，配列によるデータの並びを考慮する必要があるため，データメモリ上のデータの配置は非常に難しい問題となる．

## 3. ブロック浮動小数点システムのためのデータ配置手法

本手法では，ブロック浮動小数点表現を用いた上記の対象システムの設計において，アプリケーション中の実数データが，システムの要求する仕様を満たすために必要となる整数部および小数部のビット長が与えられたときに，データの総ビット長が最小となるデータ配置とビット長を求めることを目的とする．

また，本手法では，プロセッサベースの専用演算器を対象とする．したがって，データの仮数部が置かれるデータメモリ上の各データは，すべて等しいビット長を持つ．また，仮数部とブロック指数部の対応を容易にするため，ブロック指数は専用のメモリに格納される．

以下の記号を定義する．

- アプリケーションに含まれるデータ数：$n$
- 1ブロックあたりの要素数：$n_e$
- ブロック数：$n_b$
- ブロック指数ビット長：$L_b$
- 仮数ビット長：$L_m$
- 配置を求めるデータ：$v_1, v_2, ..., v_n$

### 入力

- データ数 $n$
- 同一配列に格納されるデータの組合せ $\mathbf{A}$
  $\mathbf{A} = \{A_j\}$
  $A_j = (v_{j_1}, v_{j_2}, ..., v_{j_k})$
- 要求される仕様を満たすためにデータ $v_i$ が最低限必要とする整数部 $v_i.iwl$ および小数部ビット長 $v_i.fwl$

### 出力

- データメモリ上のデータ配置
- 仮数部ビット長 $L_m$ および指数部ビット長 $L_b$

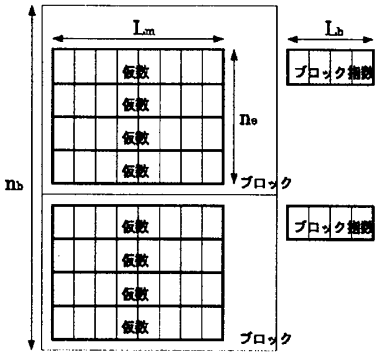


図 3: ブロック浮動小数点システムのメモリモデル

### 目的関数

データの総ビット長 $L_{all}$ は式(2)で表される．式(2)を最小化するのが目的である．

$$L_{all} = (L_m * n_e + L_b) * n_b \quad (2)$$

## 3.1 ブロック浮動小数点表現のビット長解析

提案手法では，固定小数点表現で実装するときに要求される仕様を満たすために各データが最低限必要とする整数部および小数部ビット長を入力としている．これらの値は固定小数点システムのビット長見積り手法[2]によって求めることができる．

まず，1 つのブロックで必要となるビット長を解析する手法を説明する．精度の要求を満たすためにブロック $B$ が必要とする仮数部ビット長 $Lm_B$ は，式(3)で求められる．

$$Lm_B = \mathrm{Max}_{i \in B}(v_i.iwl) + \mathrm{Max}_{i \in B}(v_i.fwl) \quad (3)$$

また，ブロック $B$ が必要とする指数部ビット長 $Lb_B$ は，式(4)で求められる．

$$Lb_B = \log_2 \mathrm{Max}_{i \in B}(v_i.fwl) + 1 \quad (4)$$

システム全体が必要とする仮数部 $L_m$ および指数部ビット長 $L_e$ は，各ブロックが必要とする仮数部および指数部ビット長の最大値であるから，それぞれ式(5)，式(6)で求めることができる．

$$L_m = \mathrm{Max}_{a \in \{1..\ n_b\}}(\mathrm{Max}_{i \in Ba}(v_i.iwl) + \mathrm{Max}_{i \in Ba}(v_i.fwl)) \quad (5)$$

$$L_b = \log_2 \mathrm{Max}_{i \in \{1..n\}}(v_i.fwl) + 1 \quad (6)$$

## 3.2 データのブロック化

提案手法は，Kernighan and Lin アルゴリズム[7]を基に，以下の点を拡張した手法である．Kernighan and Lin アルゴリズムは $n$ 個のノードをコストが最小となるように 2 個のブロックに，それぞれの要素数が等しくなるように分割するためのアルゴリズムである．

- 任意数のブロックへの分割
- 連続する複数のデータの交換

最小化対象であるコストは，総ビット長 $L_{all}$ と等しいものとする．$v_i$ が必要とする整数部ビット長を $v_i.iwl$，小数部ビット長 $v_i.fwl$ とすると，$L_m$ および $L_b$ は前節のビット長解析方法を利用し，それぞれ式(7)，式(8)で求められる．

$$L_m = \mathrm{Max}_{a \in \{1..\ n_b\}}(\mathrm{Max}_{i \in Pa}(v_i.iwl) + \mathrm{Max}_{i \in Pa}(v_i.fwl)) \quad (7)$$

$$L_b = \log_2 \mathrm{Max}_{i \in \{1..n\}}(v_i.fwl) + 1 \quad (8)$$

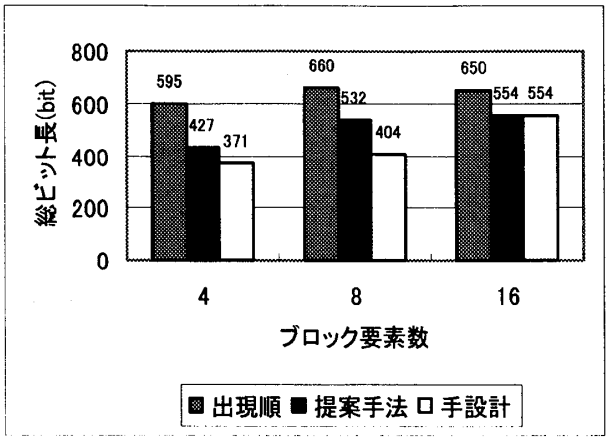


図 4: 色空間変換への適用結果

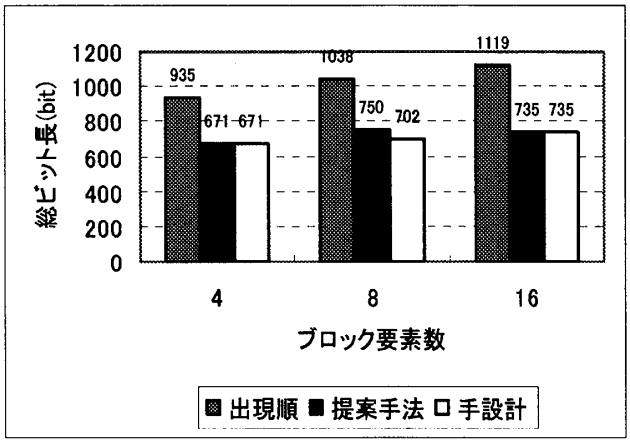


図 5: FFT フィルタへの適用結果

初期分割として $n$ 個のデータを順列として並べ，下位から順に $n_e$ 個ずつ $n_b$ 個のブロック $P_1, P_2, \ldots, P_{n_b}$ に分割する．ここから，コストを減少させるようにデータを交換することで，コストを最小化する．

配列の先頭要素および配列でないすべてのノード $v_i$, $v_j$ ($i \neq j$)の組に対して，以下のコストを式(2)に従って計算する．

- $v_i$, $v_j$ が共に配列でない場合，$v_i$ と $v_j$ を交換した後のコスト
- $v_i$ が要素数 $L$ の配列の先頭要素であり，$v_j$ が配列でない場合，$v_i$ を含む配列の要素すべてと $v_j$ から連続する $L$ 個のノードを交換した後のコスト．ただし，$v_i$ から $L$-1 個後のノードが配列の先頭以外の要素であるとき，コストは∞とする．

すべての $v_i$, $v_j$ の組のうち，コストが最小となるものを選択し，選択したデータをマークする．以後，既に交換されているものとしてコストの計算を行う．これをすべてのデータがマークされるまで繰り返す．

次に，上の繰り返しのうちコストが最小となった交換について，そのコストが交換前のコストより小さくなるなら，その交換を実行してマークを全て解除し，上の繰り返しに戻る．コストが交換前より小さくならないなら，交換前のデータ配置をコストが最小となる解として終了する．

## 4. 評価実験

### 4.1 実験内容

提案手法が実用的なアプリケーション設計の際に使用できることを確認するため，色空間変換アプリケーションおよびFFTフィルタに対して提案手法を適用した．

色空間変換アプリケーションの入力は 0 から 255 の値をとる整数値 3 個であり，出力は実数 3 個ある．色空間変換アプリケーションは配列は含まない．本実験では，出力となる 3 個の実数の誤差がそれぞれ 0.5 以内に収まるように設計する．また，ブロック浮動小数点表現を用いて設計する際，ブロック要素数は 4, 8, 16 の 3 通りで設計する．

また，FFT フィルタの入力は 0 から 255 の値をとる整数値 11 個であり，出力は実数 1 個である．FFT フィルタは配列を含み，データの約半数が配列に含まれる．本実験では，出力となる実数の誤差が 0.5 以内に収まるように設計する．また，ブロック要素数は色空間変換空間アプリケーションと同様は 4, 8, 16 の 3 通りで設計する．

提案手法により設計した結果が従来の手法により設計した結果と比較するために，提案手法により設計したシステムの総ビット長と従来の手法で設計したシステムの総ビット長と比較した．比較対象とした従来の手法は，データの出現順にデータメモリに配置する手法であり，ビット長の削減を考慮しないデータの配置手法である．

また，結果を最適なデータ配置を行った場合と比較をするために，手設計による最適解も比較対象とした．

### 4.2 実験結果

色空間変換アプリケーションに提案手法を適用した結果を図 4 に示す．図 4 より，従来手法と比較して 10-30%のビット長を削減していることが確認できる．

FFT フィルタへの適用結果を図 5 に示す．この場合も，従来手法より最大 30%程度ビット長を削減していることが確認できる．また，手設計と比較してもほぼ同等のビット長で設計できることが確認できる．

色空間変換プログラムでは，手設計の値と比較して，一部のデータでビット長が増大していることが確認できる．これは，Kernighan and Lin アルゴリズムによる最適化の限界が考えられる．

Kernighan and Lin アルゴリズムは発見的アルゴリズムであり，出力された解が最適解であることは保証されない．そのために下限より大きいビット長が結果として求められている．

本実験で配列を含む FFT フィルタの方が多くのビット長を削減できたという結果も，Kernighan and Lin アルゴリズムによる最適化の限界によるものであると考えられる．

本実験では同一配列内の各データが必要とする整数部および小数部ビット長の差が小さく，配列を多く含むデータほど，提案手法適用前のデータの配置が最適解に近い状態をとったためである．

最適解であることが保証される解を実用的な時間で求める手法の確立は今後の課題である．また，提案手法では，

データの総ビット長のみを最適化対象としているため，データの総ビット長以外の部分は考慮されていない．そこで，実行サイクル数，消費電力などのトレードオフを考慮した，総合的なデータの配置手法の確立も今後の課題として挙げられる．

## 5. まとめ

本稿では，ブロック浮動小数点システムのためのデータ配置手法を提案した．

提案手法によって，ブロック浮動小数点システムにおいて，要求された精度を満たしつつ，実数データのビット長を小さくすることができる．

提案手法では，Kernighan and Lin アルゴリズムを複数のデータブロックに分割できるように拡張している．これにより，ブロック浮動小数点システムにおける，データの配置を行うことができる．

また，提案手法は Kernighan and Lin アルゴリズムを，複数のデータを常に連続した領域を確保するように拡張することによって，配列を連続した領域に格納するデータ配置が可能である．

実験では，提案手法によりブロック浮動小数点システムのデータを配置した場合，出現順にデータを配置した場合と比較して約 10%から 30%程度小さいビット長で精度の要求を満たすシステムが設計できるという結果が得られ，提案手法が有効であることを確認した．

今後の課題は，ハードウェア量，消費電力，および実行サイクル数を対象とし，トレードオフを考慮したデータ配置最適化手法への拡張である．

## 謝辞

本研究を進めるに当たり，貴重なコメントを頂いた大阪大学今井研究室の諸氏に深く感謝いたします．

## 参考文献

[1]A.Chhabra, R.Lyer, ``A Block Floating Point Implementation on the TMS320C54x DSP,'' Application Report(SPRA610), Texas Instruments 1999.

[2]Abhijit Mitra, Mrityunjoy Chakraborty, ``The NLMS Algorithm in Block Floating-Point Format,'' IEEE Signal Processing Letters, Vol. 11, No. 3, March 2004.

[3]Abhijit Mitra, Hideaki Sakai, ``An Efficient Block Floating Point Implementation of the LMS Algorithm,'' Proc. of ICASSP 2003, Hong Kong, pp. VI--77--80, April 2003.

[4]Abhijit Mitra, ``On Finite Wordlength Properties of Block-Floating-Point Arithmetic,'' International Journal of Signal Processing, Vol. 2, No. 2, pp. 120--125, 2005.

[5]Kamen R. Ralev, Peter H. Bauer, ``Realization of Block Floating-Point Digital Filters and Application to Block Implementations,'' IEEE Transactions on Signal Processing, Vol. 47, No. 4, April 1999.

[6]Wu Jun, Hu Xiehe, CHEN Sheng, Chu Jian, ``Optimization of Block-Floating-Point Realizations for Digital Controllers with Finite-Word-Length Considerations,'' Journal of Zhejiang University SCIENCE, Vol. 4, No. 6, pp. 651--657, November 2003.

[7]土井伸洋, 堀山貴史, 中西正樹, 木村晋二, 渡邉勝正, ``Cプログラムからの合成における浮動小数点演算のビット長最適化,'' 第 6 回システム LSI ワークショップ, pp. 263--266, November 2002.

[8]B.W. Kernighan, S. Lin, ``An Efficient Heuristic Procedure for Partitioning Graphs,'' The Bell System Technical Journal, Vol. 49, No. 2, pp. 291--307, February 1970.