動画圧縮技術における高速SAD計算回路設計の一検討

今久保 彰一† 神戸 尚志‡ 藤田 玄‡‡

†近畿大学大学院 総合理工学研究科
‡近畿大学 理工学部 電気電子工学科
‡‡大阪電気通信大学 情報通信工学部

概要

動画圧縮技術における動き検出は最も重要な圧縮技術の1つである．しかし，この処理は多量の画素データを用いるためメモリの読み込み処理に時間がかかる．また，ブロックサイズごとに処理が異なるため回路面積が増大してしまう問題がある．

本論文では，各種ブロックサイズに対応したSAD計算回路と，メモリ読込回数の削減及びパイプライン化による回路の高速化を提案する．

A high-speed SAD calculation circuit design for video cording

Shoichi Imakubo† Takashi Kambe‡ Gen Fujita‡‡

†Graduate School of Science and Engineering Research, Kinki University
‡Department of E & E, Faculty of Science and Engineering, Kinki University
‡‡Faculty of Information and Communication Engineering, Osaka Electro-Communication University

Abstract

Motion detection in video compression technology is one of the most important compression techniques. However, this process takes long time for loading large amounts of pixel data from memory. Also, the circuit area is increased for supporting various block types. In this paper, a high speed SAD calculation circuit design for various block type is proposed and its circuit is accelerated by the reduction of the memory access time and by pipelining for SAD calculation.

1 はじめに

動画像圧縮技術の進歩により，H.265/HEVC [1][2](以下H.265)では，高精細テレビでは，1/160倍まで圧縮することが可能である．しかし，圧縮率を高めるほどアルゴリズムが複雑になり，解像度を上げるほど扱うデータ量が肥大化し，処理に時間がかかることが大きな問題となっており，様々な研究が行われている．[5][6][7][8]

本研究では，H.265の処理の1つである動き検出技術に対し，高速かつ様々なブロックサイズに対応できるSAD計算回路を提案する．

2 動画圧縮技術

本章ではH.265エンコーダアルゴリズムの概要と，本研究の対象となる動き検出について述べる．

なお本論文は，JCT-VC (Joint Collaborative Team on Video Coding)による参照ソフトウェアHM14．0[4](以下HM)を参考にしている．

2.1 ブロック分割

H.265は符号化効率を良くするために以下の4種類のブロック分割がある．

・CTU(Coding Tree Unit，符号化ツリーユニット)
・CU(Coding Unit，符号化ユニット)
・PU(Prediction Unit，予測ユニット)
・TU(Transform Unit，変換ユニット)

CTU はスライスを左上から右下に向かって固定ブロックサイズに分割したものである．このブロックサイズはパラメータ設定に基づいており，64×64，32×32，16×16 画素のいずれかに設定できる．

CU は CTU を再帰的に分割するブロックである．分割しない場合は CTU と同じサイズになり，最大 64×64 画素，最小 8×8 画素である．

PU は CU を分割するブロックであり，最大 64×64 画素，最小 4×4 画素である．画面内予測符号化，画面間予測符号化の基本単位になる．

TU は CU を分割するブロックであり，最大 32×32 画素，最小 4×4 画素である．PU と違い，変換処理に用いられる．

H.265 はこれら 4 種類のブロック分割を用いることにより，

1. 空など，画像中の変換が平坦な領域を H.264[3] より大きなブロックサイズで符号化することによる符号化効率の向上
2. エッジ (物などの輪郭) などの画像中の変化が大きい領域を H.264 と同レベルの詳細なブロックサイズでの符号化処理

の 2 点が両立できる．

2.2 動き補償フレーム間予測符号化

動き補償フレーム間予測符号化とは，被写体の時間的動きを利用して，参照画像から符号化対象画像を予測する技術である．画素情報をそのまま符号化するのではなく，参照画像からの動き (動きベクトル) と，動きベクトルから予想される画像 (予測画像) と符号化対象画像との差分を符号化することにより，情報量が削減できる．

2.3 動き検出

動き検出は動き補償フレーム間予測符号化に使われる動きベクトルを検出する処理である．この処理は PU 単位で処理が行われ，コスト値を用いて最適な動きベクトルを予測する．コスト値は，予測画素ブロックと符号化対象画素ブロックの差の絶対値の合計値（SAD 値）と，動きベクトルのビット量の和である．ここで言う予測画素ブロックとは，アルゴリズムにより仮定した動きベクトルを用いて予測される位置にある画素ブロックである．いくつかの予測画素ブロック毎のコスト値を求めていき，その内で最もコスト値の小さかった予測画素ブロックを参照画素ブロックに選ぶ．この参照画素ブロックから符号化対象ブロックのベクトルを動きベクトルとする．

本論文では SAD 計算のハードウェア化について考える．

3 SAD 計算回路の設計

SAD 計算は様々なサイズの PU に対して行われる．そのため HM では水平画素数毎による場合分けを行い，個別に SAD 計算を行っている．これはループ文によるオーバーヘッドの削減と，垂直画素数が 8 行を超える場合に SAD 計算対象画素数を半数にし，合計値に 2 を掛けた値を SAD 値とし，計算回数の削減が行われているためと思われる．しかし，このアルゴリズムをハードウェア実装する場合，複数種類の回路が必要なため回路面積が増大することが予想される．

本研究では，PU サイズに関係なく使用可能な SAD 計算回路を用いることで回路面積を抑え，高速化する手法を提案する．

3.1 8×8 画素高速汎用 SAD 計算回路

ハードウェア実装のアルゴリズムとして画素毎に逐次に処理する手法がある．回路面積は非常に小さくなるがオーバーヘッドが多く処理に時間がかかってしまう．一方 PU の全画素を並列処理化する手法では非常に高速であるが，ブロックサイズ毎に別々の回路を作成することによる回路面積の増大や，回路の利用効率の低下が予想される．

本研究では，PU を 8×8 画素ブロックに分割し，高速な SAD 計算回路にて計算をすることで高速かつ回路面積の増大を抑えた回路を考える．例えば 16×16 画素ブロックの場合，図 1 のように 8×8 画素ブロック 2 つに分割し，SAD 計算を行う．4 つではなく 2 つなのは，垂直画素数が 8 行を超えているため SAD 計算対象画素数を半分にしているからである．

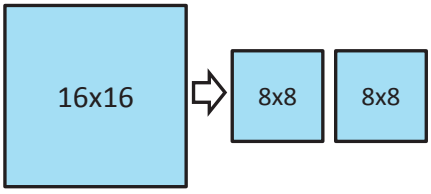

図 1: 8×8 画素ブロックへの分割

PU サイズ毎の 8×8 画素高速 SAD 計算回路使用回数を図 2 に示す．例えば 16×16 画素の PU の場合，この計算回路を 2 回使用することで SAD 値が求められる．

垂直画素数 \ 水平画素数	64	48	32	24	16	12	8	4
64	32	24	16		8			
48	24							
32	16		8	6	4		2	
24			8					
16	8		4		2	2	1	1
12					2			
8			4		2		1	1
4					2		1	

図 2: 8×8SAD 計算回数

3.2 汎用 SAD 計算回路の処理フロー

提案する回路の全体の処理フローを図 3 に示す．まず PU の水平画素数，垂直画素数，スライスの水平画素数，垂直画素数，PU 基準点 (左上の画素) のスライス上の座標をブロック情報として読み込む．これらは上位の関数からの引数にするため，内部メモリとしてテストベンチで用意する．次に水平画素数，垂直画素数により場合分けをする．PU のサイズにより分割方法などが異なる．この処理は判別のみなので 1cycle 以内で可能である．「画素読込」は計算対象となる参照スライス，符号化対象スライスの画像を外部メモリより読み込む処理である．読み込む画素の位置はブロック情報から求める．次に読み込んだ画素データを 8×8 画素ブロックに「分割」する．これは回路の配線のみなので，処理時間はかからない．分割後のデータを「8×8SAD」計算回路に渡し，SAD 計算を行う．最後に 8×8SAD の結果を合計し，PU の垂直画素数が 8 行を超える場合は合計値を 2 倍にするためのシフト処理を行なう．

3.3 画素読込・切り出し手法

図 3 の処理フロー中の画素読込ではビット連結を用いて画素読込回数の削減を行う．しかし，スライスのすべての画素をビット連結しようとすると読み込み時に必要なレジスタが膨大になってしまうため，4×4 画素の 256 ビットを 1 つのメモリブロックとし，256 ビットずつ読み込むことで回路面積と読込時間の削減を両立できる．例えば 8×8PU の場合，ビット連結を適用しない場合は 64 回の読込が必要だが，この方式を適用すると図 4 のように最大 9 回，最小 4 回のメモリ読込で済む．

しかし，メモリブロックの境界と PU の境界は必ずしも一致しない．そのため読み込んだ画素データを切り出し，PU に合わせて切り出す処理が必要と

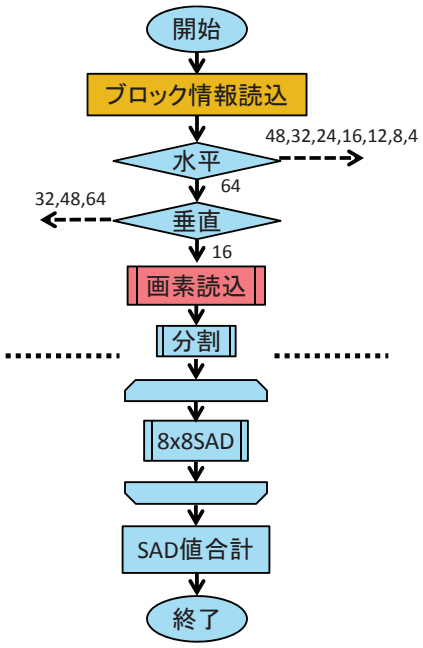

図 3: SAD 計算回路の処理フロー

なる．8×8PU の場合で，図 5 左の濃い青色部が必要な PU とすると薄い赤色部が読み込むメモリブロックと．そして左上のメモリブロックは右下 1 画素を PU の左上の画素に切り出し，上中央のメモリブロックは下 4 画素を図 5 のように PU の左上から右に 2 番目から 5 番目の画素に切り出す．同様に他のメモリブロックも切り出し処理を行う．

この画素データ切り出し処理は図 6 のように，外部メモリからの画素読み込み処理とパイプライン化をする．1 つのメモリブロックを読み終わり次第切り出しを行い，その切り出し処理中に次のメモリブロックを読み込むようにする．これにより回路面積をほとんど増やすことなく高速化が可能である．

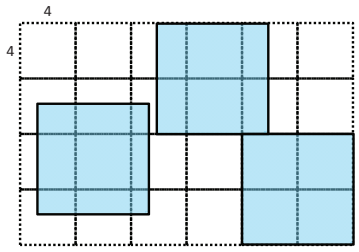

図 4: SAD 計算回路における画素読込

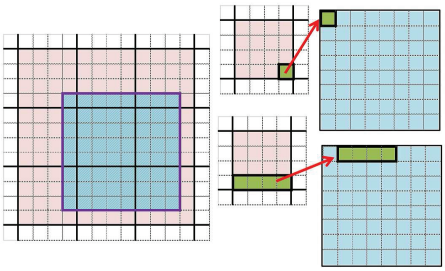

図 5: 画素切り出し

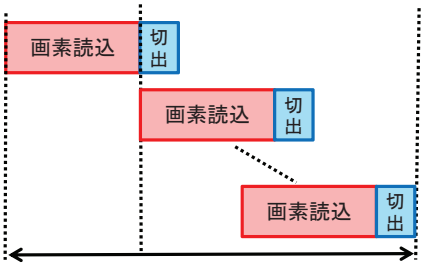

図 6: SAD 計算回路のパイプライン化

3.4 4×8 画素 SAD 計算回路

8×8 画素高速汎用 SAD 計算回路を設計する前段階として，4×8 画素 SAD 計算回路を設計する．

ここでは PU の垂直画素数が 8 行を超えるかどうかのフラグを管理する iSubShift 変数，ビット深度 (画素値の量子化ビット数) によるコスト値の変化を吸収する処理に使用する DISTORTION_PRE 変数を考慮した設計とする．また，iSubshift 変数，DISTORTION_PRE 変数，参照画素値，符号化対象画素値は外部メモリから読み込み，SAD 値を外部メモリに書き込む．

3.4.1 ビット幅削減

HM のソフトウェアプログラムでは 1bit 単位のビット幅指定ができないため必要以上のビット幅を使用している場合が多い．例えば HM では iSubShift 変数は 32bit の符号付き整数型を用いている．しかし，実際に代入される値は 0 または 1 なので，1bit で十分である．ハードウェアではこれを 1bit の符号なし整数型に指定することで 31bit のビット幅削減が可能である．このように，ビット幅の最適化を行うことで，回路面積と処理時間を削減する．

3.4.2 SAD 計算の並列化

3.4.1 項の回路ではメモリ読込や SAD 計算など，逐次に処理するが，ハードウェアでは並列処理が容易に可能である．そこで SAD 計算を並列に行うことで，処理時間を削減する．ただし，同時に計算するために同じ回路を複数生成するため回路面積は増加する．SAD 計算の並列化アルゴリズムを図 7 に示す．

まず，外部メモリから，iSubShift， DISTORTION_PRE 変数を各 1 回ずつ，符号化対象画素 piOrg，参照画素 piCur を各 32 回ずつ読み込む．図 7 の読込がそれに当たる．

次に piOrg と piCur の値の差分の絶対値 (AD 値) を計算する．図 7 の AD(Absolute Difference，差の絶対値) がそれに当たる．それぞれの AD 値をトーナメント式に合計していき，この合計値を iSubShift=1 の場合は 2 倍に，最後に DISTOROTION_PRE の値分だけ右ビットシフトした値が SAD 値である．この SAD 値をメモリに書き込む．

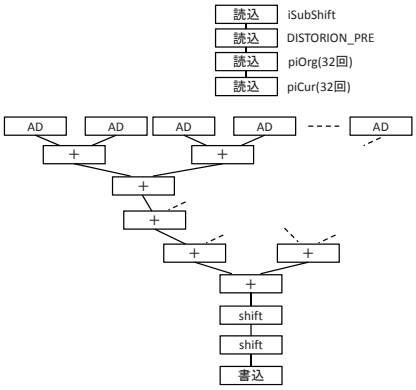

図 7: SAD 計算並列化手法のアルゴリズム

3.4.3 メモリ読込のパイプライン化

3.4.2 項の手法では，AD 計算を同時に行うため，AD 計算回路が複数必要になる．また，SAD 計算用の加算回路も同時に複数個必要になる．これらは回路面積の増加に繋がる．

回路面積の増加を抑えつつ，処理時間を削減する手法の 1 つにパイプライン化がある．今回設計する回路ではメモリ読込，AD 計算，AD 値の加算，メモリ書込などの処理をパイプライン化する．

3.4.4 ビット連結によるメモリ読込の削減

メモリ読込にかかる時間は読み込むビット数に関係なく一定である．3.4.3 項の回路では piCur，piOrg

を各 32 回ずつ読込している．そこで，それぞれの変数 32 個分をビット連結により 1 つの変数にして外部メモリに保存し，処理時のメモリ読込後に再び分割することで，メモリ読込回数と処理時間を削減できる．この手法により，メモリ読込回数が 66 回から 4 回に削減される．

3.5 4×8SAD 計算回路の設計結果と考察

本章では 3.4 節で提案した高速化手法を適用した設計結果を示し，それに対する考察をする．

HM より処理に必要なデータを 4×8 画素 PU10 ブロック分取得しテストベンチデータとした．このデータを設計したハードウェアの外部メモリに書き込み，動作時に読み込むことで検証する．また，回路設計が正しくできていることを処理結果が HM と一致することで確認する．本来，HM の仕様上 4×8 画素 PU は iSubShift=0 であるが，iSubShift=1 の場合のデータも作成し検証した．なお，クロック周波数は 100MHz，外部メモリアクセスは 50ns とした．

設計結果を表 1 に示す．

表 1: 4×8SAD 計算回路の設計結果

ver.	処理時間 [ns]		回路面積 [gates]
	iSubShift=0	iSubShift=1	
v0	5,650	5,370	22,532
v1	5,320	3,200	5,992
v2	5,470	5,470	20,666
v3	3,750	2,070	3,399
v4	370	370	12,190

3.5.1 ビット幅削減の設計結果

3.4.1 項で提案した手法を適用して設計した．これを v1(バージョン 1) とし，適用前のバージョンを v0 とする．設計結果のフロー図を図 8 に示す．

図 8 の読込はメモリ読込を，for はメモリ読込処理を for 文 (一定回数のループ) で行っているため，ループ回数の計算と判定に 10ns かかっていることを，AD ブロック 1 つは AD 計算 4 回と加算を 30ns で行っていることを，シフト読込はシフト計算とメモリ書込を 70ns 内で行っていることを示している．

ビット幅を削減した結果，回路面積は約 73%削減され，処理時間が 5%前後減少した．これは，レジスタの数や，加算や減算などを行う回路のサイズが縮小したことによると考えられる．

3.5.2 SAD 計算の並列化の設計結果

v1 に対して 3.4.2 項で提案した手法を適用して設計した．これを v2 とし，設計結果のフロー図を図 9 に示す．

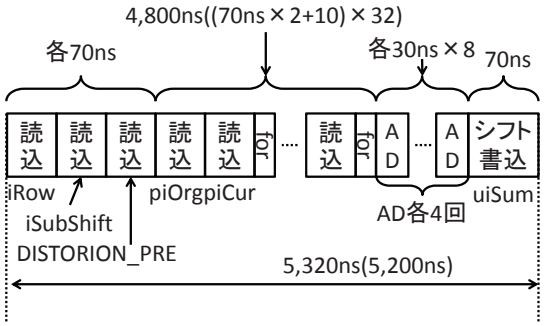

図 8: ビット幅削減の設計結果

設計結果は処理時間が v1 に比べ 13%増の 5，470ns になった．これは piOrg，piCur のメモリ読込後に他のスレッドに渡すための通信が 31 回入ったためと，AD 計算・加算の前後にスレッド間の同期通信が行われるためと考えられる．

また，回路面積が v1 に比べ 245%増加した．これは並列化のために AD 計算回路やレジスタがより多く必要になったためであると考えられる．以上のことから，この手法では十分な高速化は得られなかった．

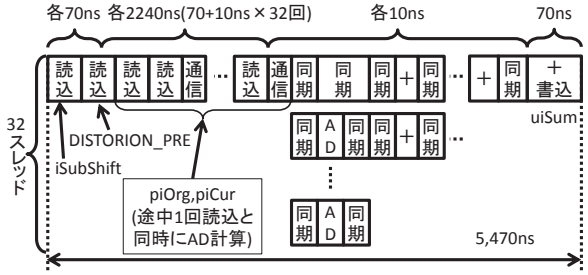

図 9: SAD 計算の並列化の設計結果

3.5.3 メモリ読込のパイプライン化の設計結果

v1 に対してメモリ読込のパイプライン化を適用した．これを v3 とし，設計結果のフロー図を図 10 に示す．

設計結果は処理時間が約 30%増加の 3，750ns (iSubShift=1 の場合 2，070ns) であった．これは for 文を用いたことによる処理回数の計算処理やこれに伴う同期通信の待機時間の発生，読み込んだ値を他のスレッドに送信する通信によるものであると考えられる．しかしながら，v1 に比べて約 30%(60%) の高速化と約 43%の回路面積の縮小が実現できた．

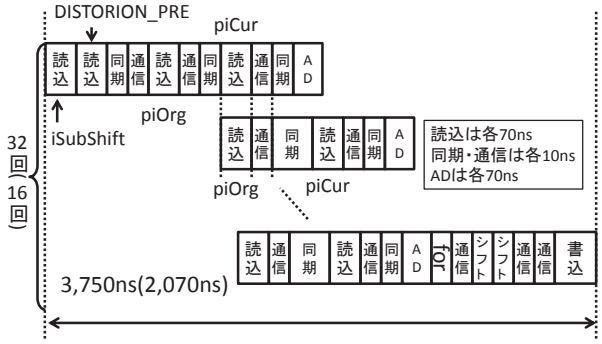

図 10: メモリ読込のパイプライン化の設計結果

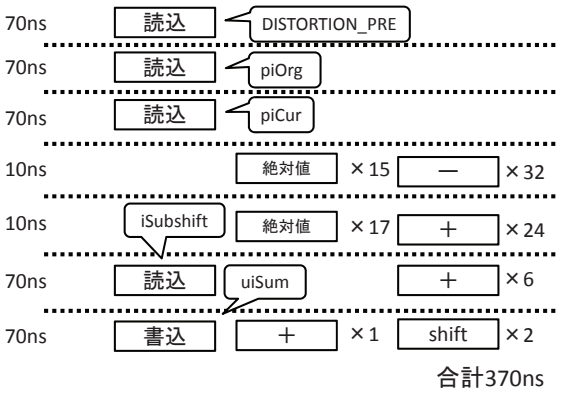

図 11: ビット連結によるメモリ読込の削減の設計結果

3.5.4 ビット連結によるメモリ読込の削減の設計結果

v1 に対してビット連結によるメモリ読込の削減を適用した．これを v4 とし，設計結果の CDFG（制御データフローグラフ）を図 11 に示す．

図 11 の上から下が時間軸正の方向であり，点線で区切られた区間が同じクロックである．例えば 4 行目で絶対値計算 15 回と減算 32 回が 10ns 以内に動作する．このように，初めにメモリ読込 3 回を逐次に行い，処理開始から 210ns 後に絶対値計算 15 回と減算 32 回を，220ns 後に絶対値計算 17 回と加算 24 回を，230ns 後に iSubShift のメモリ読込と加算 6 回を，300ns 後に結果の uiSum のメモリ書込みと加算 1 回，シフト 2 回を行っていることを示している．なお，減算と絶対値計算 1 つずつで AD 計算 1 つに等しい．

設計結果は約 45%(27%) 早い 370ns(370ns) であった．これは，AD 計算の並列化が行われたことと，加算計算が 10ns 内に複数回できたことによるものと考えられる．v1 と比較して約 93%(93%) の高速化出来たが回路面積は約 103%増加した．しかし，回路面積の増加割合より処理速度の向上の効果がはるかに大きいため，この手法は有効であると考えられる．

4 まとめ

8×8 画素高速 SAD 計算回路を提案した．また，4×8 の SAD 計算回路を設計し，性能評価を行った．

その結果，4×8 画素 SAD 計算回路ではビット連結によるメモリ読込の削減が最も効果的であり，逐次処理に比べ，約 33 倍の高速化が実現できた．今後この設計を元に，8×8SAD 計算回路の設計を行い，その性能を評価する．

参考文献

[1] 大久保榮， 鈴木輝彦， 高村誠之， 中條健 “H.265/HEVC 教科書，”株式会社インプレスジャパン，東京，2013.

[2] 村上篤道，浅井光太郎，関口俊一: ”高効率映像符号化技術 HEVC/H.265 とその応用，”オーム社，2013.

[3] 大久保榮， 角野眞也， 菊池義浩， 鈴木輝彦， “改定三版 H.264/AVC 教科書，”インプレス R&D，東京，2009.

[4] ”参照ソフトウェア HM14．0”
`https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/`

[5] Sullivan, G. J., et al. ”Overview of the High Efficiency Video Coding (HEVC) Standard.” Circuits and Systems for Video Technology, IEEE Transactions on 22(12): pp.1649-1668, 2012.

[6] Viitanen, M., et al. ”Complexity analysis of next-generation HEVC decoder，” International Symposium on Circuits and Systems (ISCAS), 2012.

[7] Jian-Liang, L., et al. ”Motion Vector Coding in the HEVC Standard.” Selected Topics in Signal Processing, IEEE Journal of 7(6): pp.957-968, 2013.

[8] Haller, M., et al. ”Robust global motion estimation using motion vectors of variable size blocks and automatic motion model selection,” 17th IEEE International Conference on Image Processing (ICIP), 2010.