修士論文

題目

高効率な直交変換処理の研究

指導教員

大野 和彦 講師

2019年

三重大学大学院 工学研究科 情報工学専攻
コンピュータソフトウェア研究室

丹後 嘉浩(416M512)

内容梗概

近年，動画像の視聴や医療分野での画像生成など，PC 上での画像処理が重要なアプリケーションとなりつつある．特に動画像の高精細化に向けた，H.265/HEVC などの高度な符号化処理や，圧縮センシングなどの信号処理の登場により，高度な二次元直交変換処理のいっそうの高速化が必要とされ，汎用プロセッサにおいてはいくつかの改良が施されている．例えば x86 プロセッサは，二次元変換を SIMD 処理などを使用して実行することはできるものの，変換処理に対する専用的な命令を定義していないため，高速な実行は難しい．そこで，この問題を解決するため当研究では，従来の SIMD 命令セットに必要な最低限の機能を追加した二次元直交変換用の計算機能を提案し，その有効性を評価した．

提案した計算機能の性能評価には，一次元直交変換の計算と，キャッシュメモリから縦方向抽出を行う実行ステップを用いた．加えて，提案した計算機能のハードウェア規模を算出するため論理合成を使用した．評価結果から，従来手法の命令を実装するための面積より小さいハードウェア規模で，主要な変換処理の全ての場合で従来の命令群よりも実行ステップ数を最低でも 18%削減できることを示した．

Abstract

Recently, the image processing which includes video watching and medical image preparation become important applications at PC. Especially, the accerelation of 2-D orthogonal transformation method is required in complicated signal processing like video coding defined in H.265/HEVC and compressed sensing, the general-purpose processor implements several operations to process complicated processing. For example, x86 processor can process 2-D orthogonal transformation with SIMD operations, but it is still insufficient because it doesn't have special instructions for 2-D transformation. Hence, in order to solve this problem, in this paper, we proposed a ALU structure that adds minimum function for speeding up to the previous SIMD operations for 2-D orthogonal transformation method and evaluated its effectivity.

In order to compare the performance for signal processing of the proposed architecture, we evaluate the number of steps of 1-D orthogonal transformation and extracting column data into cache memory. In addition, I designed the ALU structure to process instructions and calculated the hardware scale by logic synthesis. As a result, we reduced processing steps of more than 18% at the all case of mainly transformation types compared to previous method with hardware scale which is smaller than conventional hardware area.

目 次

図 目 次

表 目 次

1　まえがき

1.1　研究背景

音声解析や動画像のデータ圧縮に関連する，信号処理の広い分野で欠かすことのできない処理の一つとして，DFT や DCT などの直交変換処理が使用されている．これらの計算は，信号処理全体に占める処理量の内，大きな部分を占めている．例えば，他の信号処理技術では，少ない観測データから画像信号を再構成する，圧縮センシング手法が提案されていて，それに DFT が使われている．この技術は MRI 画像生成など医療分野で活用されている [1] が，単一の画像を生成するために膨大な計算コストが生じる問題がある．DFT よりも画像処理に対する圧縮効率が優れている DCT 変換においては，処理量の割合は動き探索の次に大きく，全体の二割程度を占めている [2]．そして近年，動画像符号化の分野では，スーパーハイビジョン放送の開始などに伴って，情報量の多いデータを扱う事例が多くなり，高効率な圧縮技術が要求されている．その要求に応えるため，JCT-VC（Joint Collaborative Team on Video Coding）により従来手法の 2 倍にまで高めた H.265/HEVC[3] が 2013 年に標準化された．しかし処理内容の複雑化により，従来より DCT の処理量が増加している．これらの二次元直交変換の実現には処理量の大きい転置が一般に利用されている．従来のプロセッサでは列単位の並列処理に対応できないため，二次元直交変換に行単位の並列処理のみによる実行が強いられるのに対し，転置処理はそれを直接的に実現可能とするからである．従って，高速化方法を探るには，その要で並列処理による高速化が困難な転置の高効率化の追求が必要になる．

汎用プロセッサに搭載されている SIMD 演算命令セットでは，配列内のデータ入れ替えにはシャッフル演算，行列計算には並列乗算や並列加算が使われる．また，符号付きと符号なし演算，int 型データサイズや short 型のデータサイズなど，用途別に異なる命令が定義されている．これらの命令セットで，動画像符号化の内最も処理量が多い動き検出に使われる SAD 演算に対しては，高速化する専用命令が定義されているが，直交変換に必要な処理に対する専用命令は定義されておらず，前述の転置を含め，高速化が不十分である．転置の高速化については，32 ビット浮動小数点のデータを，高速に転置する命令群が，Intel プログラムリファレンス [4] に記されている．一次元直交変換は，逆 DCT 変換を行う命令コード [5] にあるように，様々な直交変換を SIMD 演算で高速化する試みがな

されている．

しかし上に挙げた転置方法は，複数のレジスタが持つ行方向データを全て，列方向のデータに再編させるため，命令ステップ数が多い．また，一次元直交変換処理では，レジスタ内のデータ移動に伴うシャッフル演算野命令ステップも多くなる．これらの問題が，二次元直交変換処理の高速化に向けるボトルネックとなっている．

それに対し当研究室では，動画像符号化の高速化を目的としたプロセッサアーキテクチャ[6] の検討を進めるとともに，転置処理を不要とする行・列両方向アクセスに対応するキャッシュメモリ [7] を提案してきた．ただしこのキャッシュメモリを使用するだけでは，直接的に転置を高速化することができないため，ロードしたデータの整列機能が追加で要求される．加えて，二次元直交変換の主な処理である一次元直交変換処理の高速化を実現するため，最適な処理方法を探る必要がある．

1.2 研究目的

二次元直交変換は，行列両方向の一次元直交変換と，その間の転置演算などの縦方向データ抽出処理によって構成される．変換処理を高速化する手法は，DFT や DCT などを高速にするアプローチが今までの研究にいくつか存在し，代表的な手法に，FFT や FDCT などの高速化アルゴリズムがある．ハードウェアの改良による高速化では，専用的なハードウェアを特別に設ける研究が多く，転置高速化のため特殊なバッファなどを設ける手法 [8] が一例として挙げられるが，コストに見合う使用率が得られるとは考えられないため，汎用プロセッサには適さない．対して，汎用プロセッサで定義されているベクトル演算を使った転置高速化は，二次元直交変換の大幅な削減には繋がらないが，直交変換の代表的なものである FFT や高速 DCT の入れ替えが従来命令の組み合わせで実現できる．そのため，その機能を効率化した最小限のハードウェアを追加するだけで，符号化に使用される直交変換と転置演算などの汎用的な処理能力が向上する可能性がある．

従来の直交変換処理は，専用的な入れ替えがたびたび使われる直交変換に対して，汎用的な入れ替えを流用しているため，転置あるいはデータ並べ替えのオーバーヘッドが高速化のネックとなっている．そこで，転置と一次元直交変換に使われるシャッフル機能の向上を図り，その改良を

生かした高効率の二次元直交変換処理法を明らかにする．

当研究では，汎用プロセッサに搭載されている SIMD 演算命令に，直交変換向けのシャッフル命令を提唱し，従来手法の演算より高速な二次元直交変換を，最小限のハードウェア規模追加で可能にする計算機構を提案する．その評価には，二次元直交変換処理に関し転置を用いない従来法，転置を用いない行・列両アクセス対応キャッシュを用いる方法で比較する．

2　直交変換とその演算量

2.1　DFT や DCT の直交変換

一般的に，直交変換とはある情報を持ったデータを周波数成分に変換することをいい，特定の信号処理で得られる数値に数学的な係数を掛け合わせる操作を行い，信号値を周波数領域に変換する．その主な種類には，DFT（Discrete Fourier trasnform：離散フーリエ変換）や DCT（Discrete Cosine transform：離散コサイン変換）などがある．

DFT は，離散化されたフーリエ変換であり，デジタル信号の周波数解析に使用される．また，偏微分方程式やたたみ込み積分を効率的に計算する時にも使用される．DFT の式は，一般的に以下のように定義される．

$$X(k) \quad = \quad \sum_{n=0}^{N-1} x(n) \exp(\frac{-j2\pi kn}{N})dt$$

DCT は DFT のような，離散信号を周波数領域へ変換する方法の一つであり，動画像符号化などの画像処理に広く使用されている．DCT の式は，一般的に以下のように定義される．

$$X(k) \quad = \quad \frac{1}{2}x_0 + \sum_{n=1}^{N-2} x_n \cos(\frac{\pi}{N-1}nk) + \frac{(-1)^k}{2}x_{N-1}$$

この DCT 計算は動画像符号化や復号化に占める処理量が多く，ボトルネックの一つになっている．

また，代表的なこれらの二つの直交変換にかかる演算量は，$O\left(n^2\right)$ に達する．

2.2　直交変換の高速化アルゴリズム

これらの直交変換には，図 2.1 と図 2.2 に示す，高速化アルゴリズムが存在する．例えば DFT では，FFT（fast Fourier transform）が存在し，それを計算に適用することで，一次元 DFT の場合，$O(n^2)$ から $O(nlog(n))$ に計算量を削減できる．

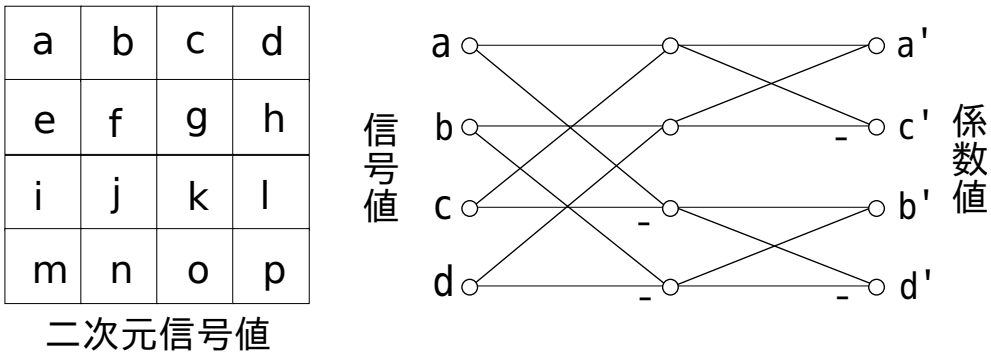

図 2.1: 4 × 4 信号値に対する FFT バタフライ演算

DCT 変換では，図 2.2 の Chen が提唱した高速化アルゴリズム [9] を使い，32 × 32 データを一列ずつ計算することで，通常の行列演算と比較して乗算と加算回数を半分程度削減できる．

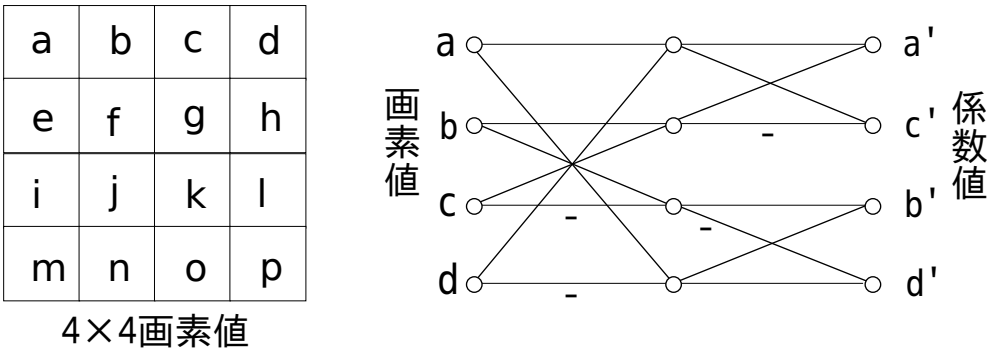

図 2.2: 4 × 4 信号値に対する DCT バタフライ演算

2.3　二次元変換処理

二次元直交変換処理は，画像データなどの圧縮などに代表される信号処理の分野で，最も多く使用されている．この二次元変換は計算量が，$O(n^3)$ に及び，一次元直交変換より計算量が増大するために，ボトルネックの一つとなっている．

2.3.1　画像符号化における変換

画像処理における DFT や DCT は，変換係数を作成するために行われる．この変換係数とは，画素値に対して二次元直交変換を行うことで，得られる値を指す．

変換係数は，図 2.3 で示す画像の中にある周波数成分を数値で表現している．この直交変換によって得られた成分に対して画像解析を行ったり，人間の目では認識しにくい高周波成分に該当する値を削減して，符号化処理に役立てるというような代表的な用途がある．

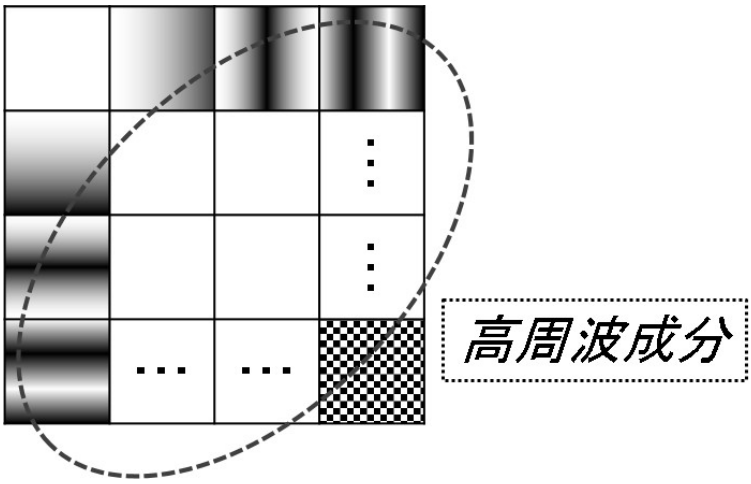

図 2.3: 画像の周波数成分

図 2.4 に示すのは，画像符号化で最も使用されている DCT 変換によって得られた係数を，4 × 4 サイズの画素ブロックに対して行った例である．この得られた DCT 係数に対し，行列内の高周波成分に該当する値を省略することで，画像データの圧縮を実現している．

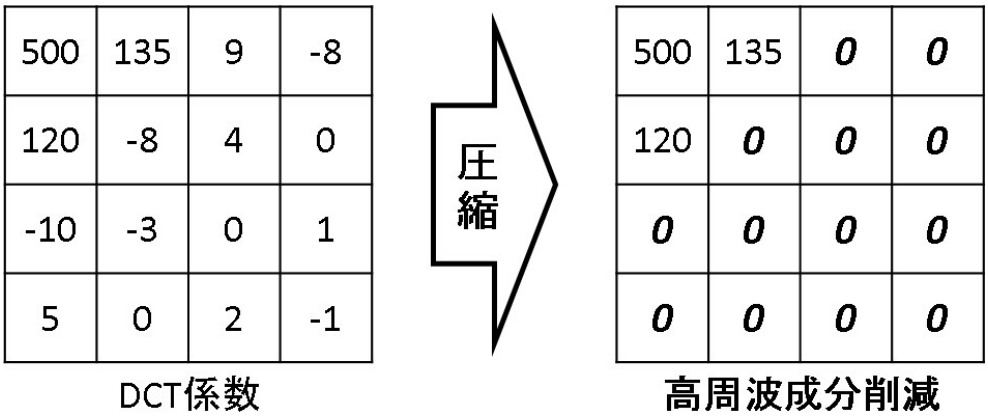

図 2.4: DCT 係数が示す意味と役割

動画像符号化では，動画を構成する 1 フレームの画像が，図 2.5 で示す変換ユニットに分割される．そして分割された領域の全部に変換処理を行うため，演算量が膨大になる．

2.3.2 H.265/HEVC の変換処理

従来規格された H.264/AVC では，4 × 4 または 8 × 8 の画像を構成する変換ブロックに対して，整数精度の擬似的な DCT 変換を使用している．

対する新しい規格の H.265 では，H.264 と比較して DCT 変換に使用する DCT 基底パターンが改良され，後で行われる量子化で必要な周波数成分ごとの正規化処理が不要になる特徴がある．

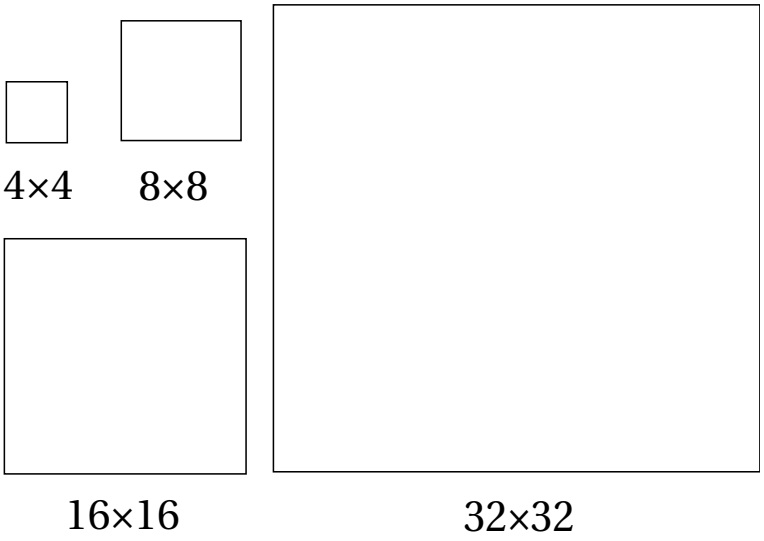

図 2.5: H.265/HEVC が定義している変換サイズ

H.265/HEVC では，変換サイズが図 2.5 に示す最小 4 × 4 から最大 32 × 32 までの 4 つが定義されている．それに伴い，DCT 変換を行うために 32 × 32 のサイズにもなる大きな DCT 係数行列をサポートしている [10].

また DCT 係数行列に，16 ビット整数値を使用することで，従来の H.264/AVC よりもさらなる変換処理高速化を図っている．

3　直交変換をハードウェア実装する際の課題

二次元直交変換を汎用プロセッサで行う場合，その機能の中で一般的な，lw や add 命令などの，スカラ命令を用いた演算や，画像処理などに含まれる大規模なデータに対して並列処理を行う SIMD 命令を用いた演算などを使い，変換行列を求める．

3.1　複雑なデータ並び替えへの対応

二次元変換を構成する一次元変換を SIMD 命令により並列処理する場合，加算と乗算の信号値計算と並び，レジスタ内で値を入れ替えることが共に必要になる．そのため高速化には，その並び替えを効率よく行えるシャッフル命令が不可欠である．

加えて，二次元直交変換を SIMD 命令による並列処理をする場合，行方向と列方向で同じ効率の SIMD 対応算術演算を行う場合，新たにレジスタに列方向データを格納する処理が必要となる．仮に，行・列両アクセス対応キャッシュなどの特殊なハードウェアが用意されている条件があれば，列方向データの抽出は，アドレス操作での対応が可能になる．従来の x86 プロセッサのように，そのハードウェアを備えていない場合，列方向データを抽出するための転置操作が要求される．汎用プロセッサに搭載されている SIMD レジスタを使用し，DCT などの直交変換を行う場合，通常の行列計算と同様に，ベクトル積和演算命令などを使用して，定義されたサイズの係数値を求める．

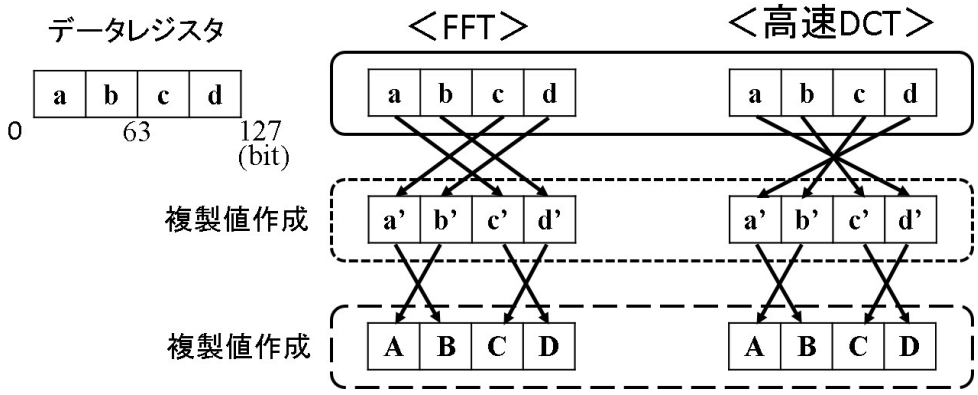

図 3.6: バタフライ演算のために行われるレジスタ内のデータ入れ替え

ただし，2 章で解説したバタフライ演算に基づいた変換がプログラムなどで定義され，それを計算機を使って行う場合，高速アルゴリズムによる乗算と加算命令の減少が期待される．その代わり，図 3.6 で示すような，レジスタ内のデータ配置を置き換えた信号値の複製データを計算に使うための，シャッフル演算が追加で発生する．

3.2 SIMD を使った高速 DCT 変換の実現

DFT を高速化した FFT 演算は，データの移動が対照的で，比較的単純な計算である．一方，図 3.7 で示す高速 DCT 変換は，FFT よりもバタフライ演算の構造が複雑であるため，直交変換の高速化を目的とする SIMD 処理を用いるためのデータ整列に，高度なシャッフル演算が要求される．

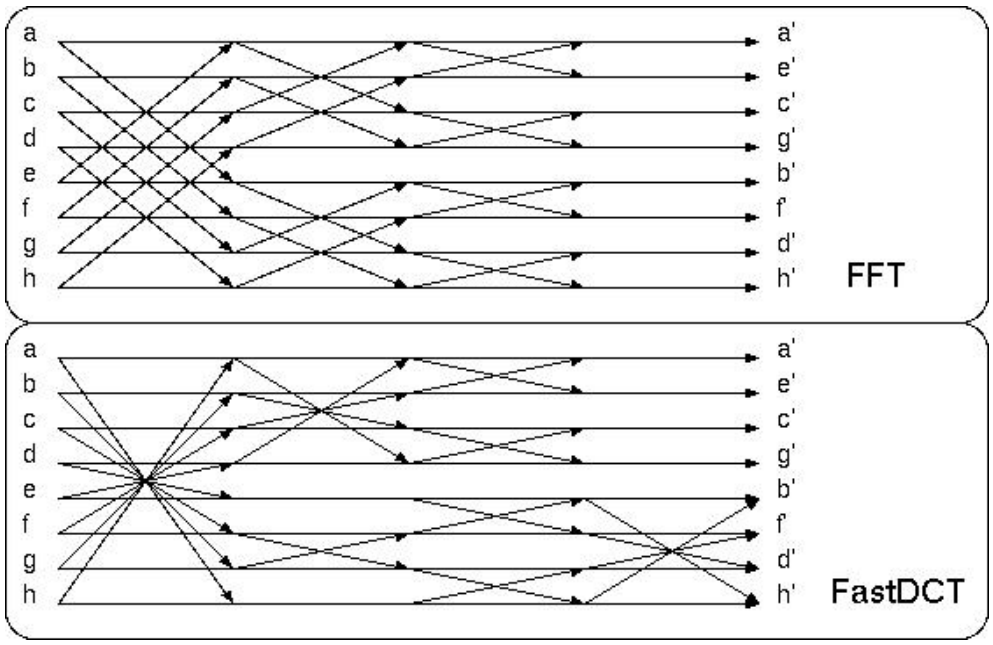

図 3.7: FFT と高速 DCT のバタフライ演算

図 3.7 で示すようにバタフライ演算の規模は行列が大きくなるほど複雑になるが，FFT の場合は行列のサイズにかかわらず，上位の値と下位の値の入れ替えが対照的に表され，高度なシャッフル演算を必要としない特徴がある．対して，高速 DCT の場合は非対称な並べ替えが多く，図 2.2 の 4 点 DCT が図 3.7 に示す 8 点 DCT に含まれているなどの特徴があるものの，基本的には行列が大きいほど入れ替えパターンが複雑になる．このため，高速実行を実現するためには，高度なシャッフル機能を定義する必要がある．

この理由により，SIMD 処理を直交変換の高速化に活用する場合，高速 DCT などの一部の計算を採用したときに発生するデータ入れ替えを効率よく行うことができるかどうかは，シャッフル命令の性能に左右される．

3.3 SIMD 命令を使った実現例

3.3.1 IntelAVX2 の SIMD 命令セット

SIMD 命令セットの代表である，IntelAVX2 命令群を使用して高速 DCT 変換を行う場合，高速 DCT 変換を行うためのデータを複製する時に，多種類のシャッフル命令を組み合わせて使用する．

主な命令では，レジスタにロードされた 16 ビットデータを対象にシャッフルを行う，vpshufhw 命令や vpshuflw 命令があるが，256 ビットレジスタにあるデータの内半分のデータ対してのみシャッフルを行うため，直交変換を高速化するための大きなボトルネックになってしまう．

3.3.2 IntelAVX512 の SIMD 命令セット

先の命令セットの改良版である，IntelAVX512 命令セット [11] では，AVX2 の命令群と比較してシャッフル演算の機能が向上した，図 3.8 の命令が定義されており，AVX2 のシャッフルを使用した場合よりもシャッフルのステップ数を一定量削減することができる．

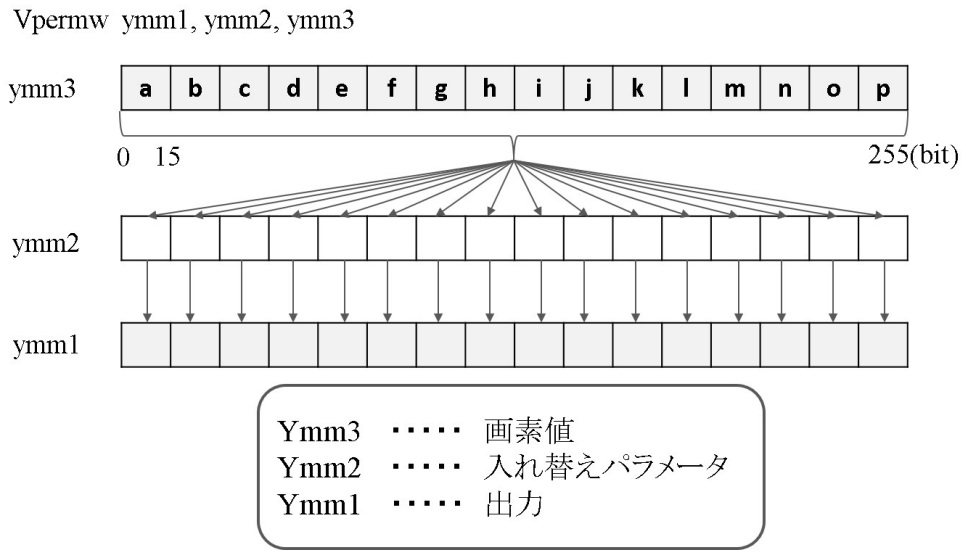

図 3.8: vpermw のシャッフル機能

しかし，この図 3.8 で示す vpermw 命令のような高機能なシャッフル演算は，IntelAVX2 の vpshufhw 命令などの従来のシャッフル演算よりも，

1 サイクルあたりの発行可能命令数が少ない [12]．そのうえ，命令の第 2 ソースで指定するレジスタにシャッフルパターンをロードする必要があり，その分追加の実行ステップが要求されるため，IntelAVX2 で行う変換処理を多少高速化するに留まっている．

3.3.3 x265 の DCT 実装

H.265 の動画像符号化処理を適用したオープンソースのプロジェクトに，x265[13] がある．この x265 には H.265/HEVC の DCT 変換処理が C++ で実装されている．変換処理を行う関数内では，全ての変換ブロックサイズに対して，DCT 変換のバタフライ演算が実装されている．これに含まれる一部の積和演算が並列演算可能な形式（例：＜ A × B+C × D ＞ → 複数のかけ算を並列実行可能）で実装されているため，一部の計算に SIMD 演算などを利用できる．

しかし x.265 では，図 3.7 のバタフライ演算によって得られる行方向係数の値 a’〜h’ を，一つずつ逐次的に求め，列方向用配列の任意の場所に格納している．このため転置処理は不要になる反面，並列演算を採用していないため，さらなる高速化の余地があると考えられる．

3.4 転置処理の複雑性

二次元直交変換は，縦横の両方向計算が必要な上，方向毎にベストの効率を得ようとすると，列方向あるいはタイル単位アクセスが利用できない場合，二方向の変換処理の間にデータの転置処理を挟む必要がある．しかしこの転置処理はオーバーヘッドが大きいため，方向毎にベストの効率を得ることが最終的に高速化に繋がるとは限らない．

転置処理に伴うデータ移動を説明するため，一つのレジスタに列方向データを集める方法を紹介する．まず従来の汎用プロセッサがキャッシュラインに対して必要な縦方向データを得る場合，レジスタ間のデータシャッフルを繰り返すことで，図 3.9 に示した列データを得る．

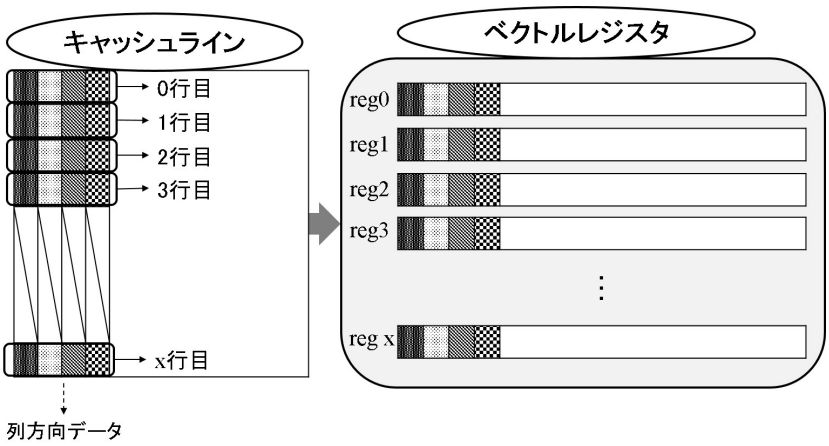

図 3.9: 縦方向データへのアクセス

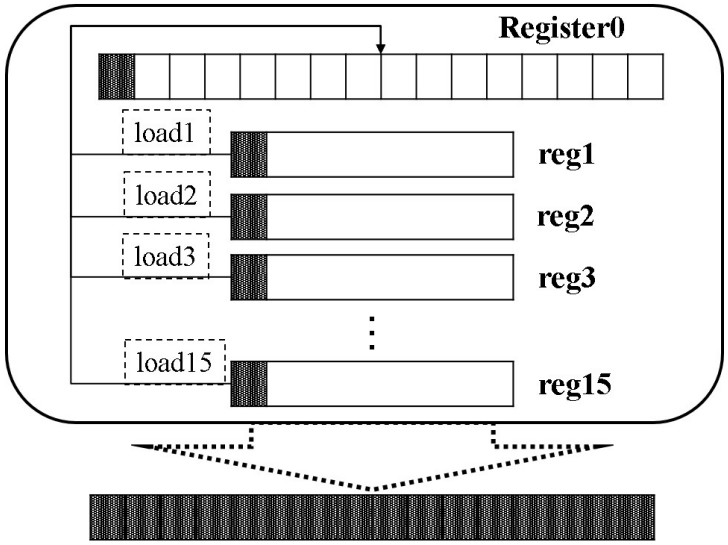

図 3.10: レジスタ 0 への列画素格納

この図3.9が示すベクトルレジスタ内のデータは，画像に対する列方向のデータに対して，従来のキャッシュアクセス方法を用いて格納されたものであるが，図で示すように，実際はこの一列のデータが異なるレジスタ間にまたがって配置される．そうして実際の転置処理を行う際には，図3.10のような，列データを集合させる処理が内部的に行われている．

キャッシュラインに対するこのようなデータアクセスは，二次元直交変換が必要とする転置処理が，図3.10のような実行ステップの多いデータ入れ替えを含む．このため，SIMD命令群を使用しても，十分な速度向上効果は得られない．従来の転置は，このようなキャッシュへのラインアクセスを行って得られた列方向のデータを，一つのレジスタにまとめるために，レジスタ間の値入れ替え命令を多く実行している．

3.5　直交変換処理の高速化に対する要求条件

汎用プロセッサにおいても，直交変換のための行列計算の高速化に向け，いくつかの命令群を提案しているものの，高速化の程度は十分ではない．理由としては，高速な周波数変換処理に向けた以下の要求を満たせていないためである．

・二次元直交変換に含まれる転置処理の高速化

・直交変換処理を対象とした専用シャッフル命令の定義

4　SIMD 演算機能の拡張案

本研究では，汎用プロセッサが行う符号化や信号処理で使用される二次元直交変換を高速化するため，最小限のハードウェアコストという条件を満たしつつ，高性能な計算機能の実装を目指す．そのためにまず，キャッシュへのタイルアクセスなど，高速化のための特別な機能を持った処理を行うため，SIMD 演算機能を持つ当研究室考案汎用データパスをベースにして，二次元直交変換の高速化を達成する計算機能を実装する．

4.1　従来の二次元変換処理

H.265 の動画像符号化を実装している x.265 では，行方向変換で求めた係数を逐次的に列方向配列に格納している．この方法は転置不要になるが，値を求めるための内部処理は，画像処理に対する処理速度が遅い逐次処理で行っている．

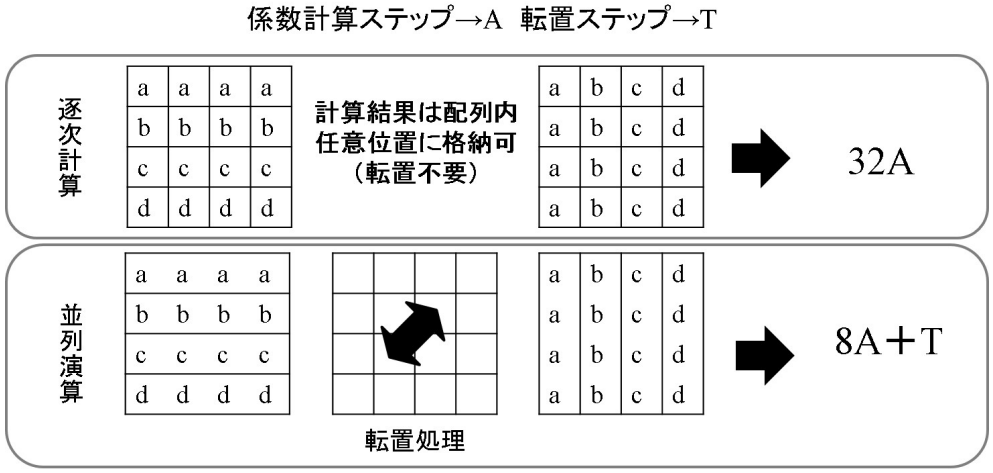

図 4.11: 逐次計算と並列演算の計算効率

図 4.11 では逐次計算と並列演算で，直交変換を行う際の計算ステップを基準とした処理時間を，単純に表している．逐次計算は行方向の変換処理を行った計算結果を，配列内の任意の位置に格納できることで，次の列方向の変換処理を転置を用いたデータ移動なしで実現できる特徴がある．

それに対して並列演算は，行方向と列方向の変換係数を一ライン毎に求めることができ，変換処理の効率が良い．しかし，多くの場合ベクトルレジスタの値を配列の任意の位置に格納できないため，列方向の処理に繋げるには，転置処理やそれに準ずる縦方向のデータ抽出を行う必要がある．図 4.11 が示す二つの処理ステップの差は，逐次計算が 24A だけ係数値計算ステップ分多く，並列演算は転置演算の T の計算ステップ分，理論上は多くなる．並列処理は変換処理の計算ステップを削減できるが，逐次計算の効率を上回るためには，転置をはじめとした縦方向データの抽出にかかる処理ステップが十分に小さいことが要求される．

4.2 転置処理の高速化

しかし 3 章の 3.4 で述べたように，従来の x86 アーキテクチャで使用されるキャッシュへのラインアクセスは，行列内の異なる列データにアクセスできないため，転置のためのシャッフル操作が多く発生し，そのための処理ステップが増加し，逐次計算より計算が速くなる条件を満たす可能性が低くなる．そこで，効率よく列データを取得するため，図 4.12 で示す，64[bit] ×縦方向 4 列のデータに対するアクセスが可能なキャッシュメモリ [7] を用いる．なおかつそれを使用して，必要な列方向のデータを効率よく抽出し，従来の SIMD 演算に必要な転置処理よりも比較的少ないステップで実行することを目指す．

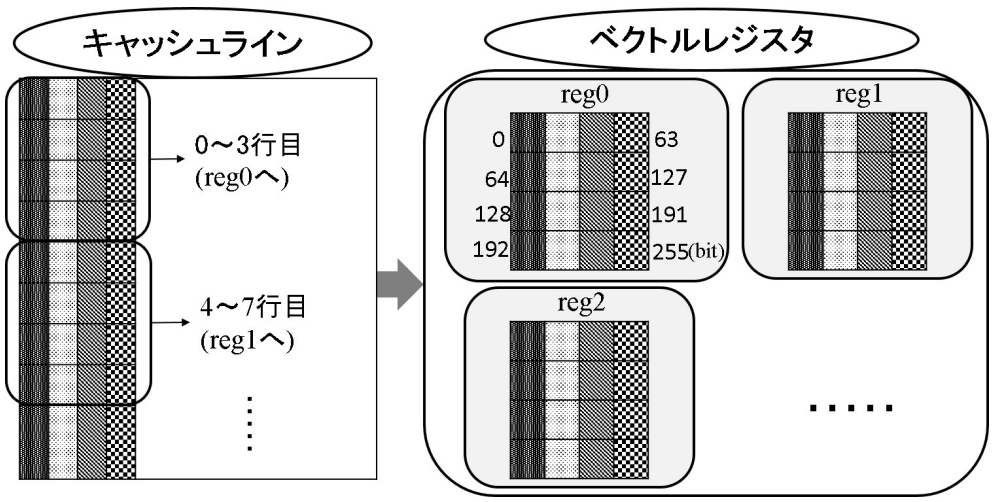

図 4.12: タイルアクセスを使用した列画素格納

列方向データが必要となる演算に対して，この図 4.12 が表すタイルアクセスを行うロード命令を使うことにより，図 3.10 のデータ格納操作の回数を削減することが期待される．

4.3 直交変換の高速化

従来手法の直交変換プログラムでは，多くの場合バタフライ演算が実装されている．そのバタフライ演算を行うための配列データのシャッフル方法には，アドレス操作によるシャッフルと，ベクトル演算を使用したシャッフル命令を活用する方法がある．アドレス操作のシャッフルにはアドレス計算が必要になるが，そのオーバーヘッドはスカラ演算部とのスーパースカラ処理により隠蔽できる可能性がある．一方 SIMD 処理を初めとしたベクトル演算でのシャッフル命令は，データ入れ替え命令と積和 SIMD 演算とがスーパースカラ実行できない場合，高速化を達成するためにデータ入れ替え命令を最小限にする必要がある．

直交変換の内，FFT の高速化アルゴリズムは比較的単純なため，高速化を念頭に置いて従来の AVX 命令セットを使用することで，十分な性能を発揮できる．しかし二次元 DFT では転置計算が，高速化のオーバーヘッドになる [14].

DCT 変換では，Chen の高速化アルゴリズム [9] が定義する計算に対し

て，SIMD 演算などを使用する場合，算術演算の合間にデータを入れ替える高度なシャッフルが必要になる．このデータを入れ替えるパターンは数通りあり，H.265/HEVC に定義されている 32 × 32 変換サイズの操作では，14 個存在する．このデータ入れ替えのパターンが DCT 高速化にとって重要となるため，各パターンに対応するシャッフルが定義できれば，最大限の高速化が実現できる．よって提案手法では，DCT 高速化アルゴリズムのフローに沿って円滑に計算できるように，従来の SIMD 命令セットにあるシャッフル機能よりも，更に DCT 変換に向いた，専用シャッフルを持つ計算機構を提案し，一次元 DCT 変換の高速化を実現する．

4.4 提案する演算機構の構成

本研究では，この章で述べている要求を満たすことができる，以下の演算機構を提案する．なおこの研究で定義する，SIMD 命令を実行する基本的なハードウェア構成に，ベクトルレジスタ，キャッシュデコーダー，SIMD デコーダーと ALU 制御などがあり，これら全機能の解説は，ベースとなるデータパスの設計を述べた論文 [6] で行っている．また回路構成についても，提案されている機能と同様のものを使用するため，以下の説明では二次元直交変換のために改良を加えた部分を解説する．

4.4.1 列方向データの抽出

二次元変換に必要な列方向データの抽出には，その高速化のために完全な転置処理を用いず，キャッシュラインにある変換処理に必要な列データのみを，図 4.12 のようなタイルアクセスを使用して取得し，少量のシャッフル演算で列方向データを抽出する．

タイルアクセスによって取得された画素値などの信号データは，5 章で述べる列方向 16 ビット抽出命令によって，一列のデータが一つのレジスタに集約され，行方向と同様の算術演算を，列方向でも計算できる形状にする．この方法によって，本来転置演算に必要となるレジスタ間データの入れ替え回数削減に繋がると同時に，転置よりも使用するベクトルレジスタを少なくし，実装の簡略化に貢献できる可能性がある．

4.4.2　専用シャッフル演算の定義

DCT 高速化アルゴリズムを用いることによって生ずるシャッフル演算を効率化するため，直交変換の高速化フローの各段階で必要な入れ替えパターンに分割し，全パターンに対応する DCT 変換専用の命令を新たに定義する．

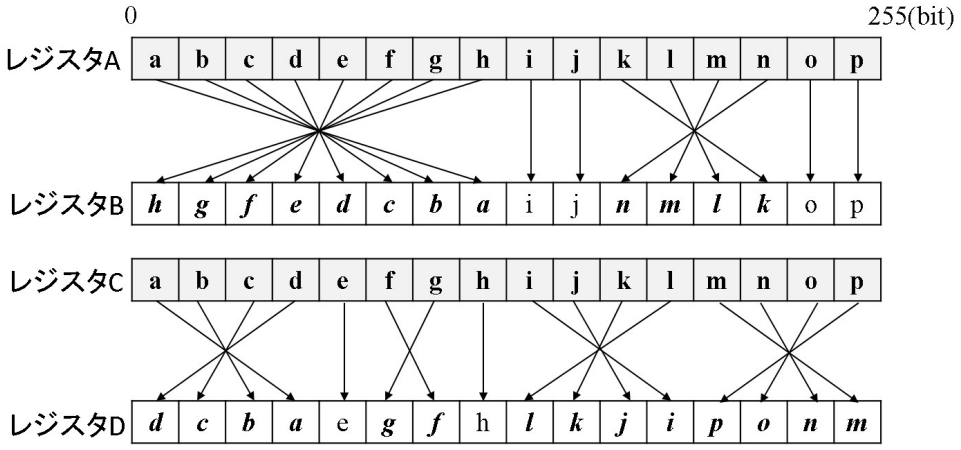

図 4.13: シャッフルパターンの例

32 × 32 の DCT では，図 4.13 で示すようなデータシャッフルのパターンが，全部で 14 通りある．これらの演算を効率化するため，それに対応する計算機能を提案する．

シャッフル計算に使用するベクトルレジスタは 256 ビット幅を使用し，これを最大 32 × 32 行列の直交変換に使用する．

この命令を実現するため，[6] で示している SIMD 演算機構の基本的な入出力とは別に，新たに追加する数ビットのシャッフルパターン指定を行う入力によって制御することで，図 4.13 の入れ替えを実現する．

5　二次元直交変換に使用する命令

二次元直交変換で使用するシャッフル計算に関連する，主な SIMD 演算の命令を紹介する．まず比較対象となる IntelAVX2 と IntelAVX512 の一部のシャッフル命令，そして提案手法の命令を順に紹介する．提案手法の命令は，ハードウェア規模低減のため，高速な直交変換の実行に必要となる最小限の命令に限定した．

5.1　従来命令セットでの構成

5.1.1　シャッフル命令群

IntelAVX2 で定義されたシャッフル命令群は，主に，レジスタの上半分をシャッフルする vpshufhw 命令と，下半分をシャッフルする vpshuflw 命令の 2 つを中心に構成される．このほかに，十種類程度の命令を追加で使い，直交変換の高速化アルゴリズムに定義されている入れ替えパターンを実現している．

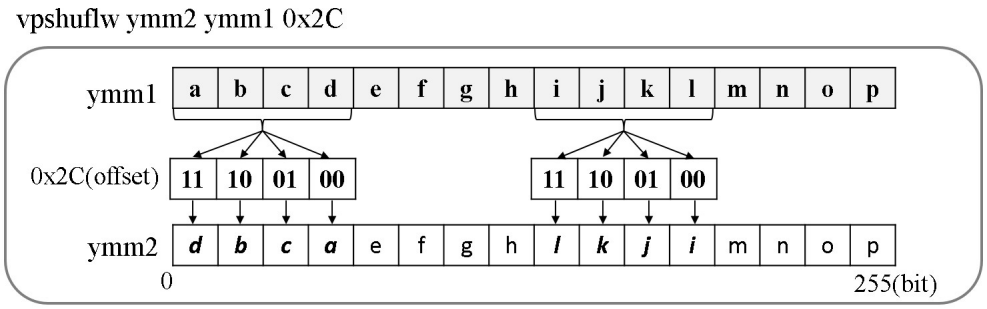

図 5.14: vpshuflw 命令

この図 5.14 で示した vpshuflw 命令は，128 ビットデータを境にして，下位の 16 × 4 データを命令の第二ソースで指定したパラメータを参照して入れ替えを行っている．この命令とよく似た命令に vpshufhw 命令があるが，そちらは上位の 16 × 4 ビットデータを入れ替える点で異なっており，二つの命令を組み合わせることによって，256 ビットのベクトルレジスタ内全体の，データ入れ替えが実現されている．しかし，この命令は

図 5.14 に示すとおり，ベクトルレジスタ内のデータに対するシャッフルは半分のデータしか，演算対象にとることができない．

5.1.2 vpermw 命令

IntelAVX512 で定義された vpermw 命令は，256 ビットのレジスタにある全ての 16 ビットデータを，任意のレジスタ内の配置に出力できる命令である．この命令の機能は，3 章の図 3.8 で示している．この命令が vpshuflw 命令と異なる点は，命令の第二ソースに別のレジスタの値を参照していることが挙げられる．この第二ソースで指定されたレジスタにパラメータを指定することで，第一ソースにある全ての 16 ビットデータは，デスティネーションで指定したレジスタ内の 16 ビットで区切った場所の内，任意の位置へ配置できる．

ただしこの命令を実行するにあたって，別の 256 ビットレジスタにパラメータとなる値を設定しなければならないために，別にロード命令が要求される．また，この命令のスループットが 2[cycle/instruction] となっているため，他のシャッフル命令の二命令分の実行ステップを要する．

これらの理由があるため，全てをこの命令で実行するのではなく，場合によっては IntelAVX2 の命令群を使用した方が，実行ステップが少なくなる．

5.2 提案命令セットの構成

提案する命令セットは，ハードウェア規模低減のため，二次元直交変換の実行に必要な最小限のベクトル命令に限定する．

5.2.1 TRANSPOSE16BIT 命令

TRANSPOSE16BIT 命令は，図 4.12 のキャッシュライン内のデータで示すような，列方向に並んだ 16 ビットデータの抽出を行うため，複数のレジスタ間で順番にデータ交換を行う命令である．

図 5.15 に示すように，タイルデータを持つ 4 つのレジスタに対し，レジスタ内のデータを半分ずつ交換する．この命令を繰り返すことによって，一つのレジスタに一列のデータが抽出できる．この命令の第三ソースのパラメータを，1 列目と 2 列目の交換または 1 列目と 3 列目の交換といっ

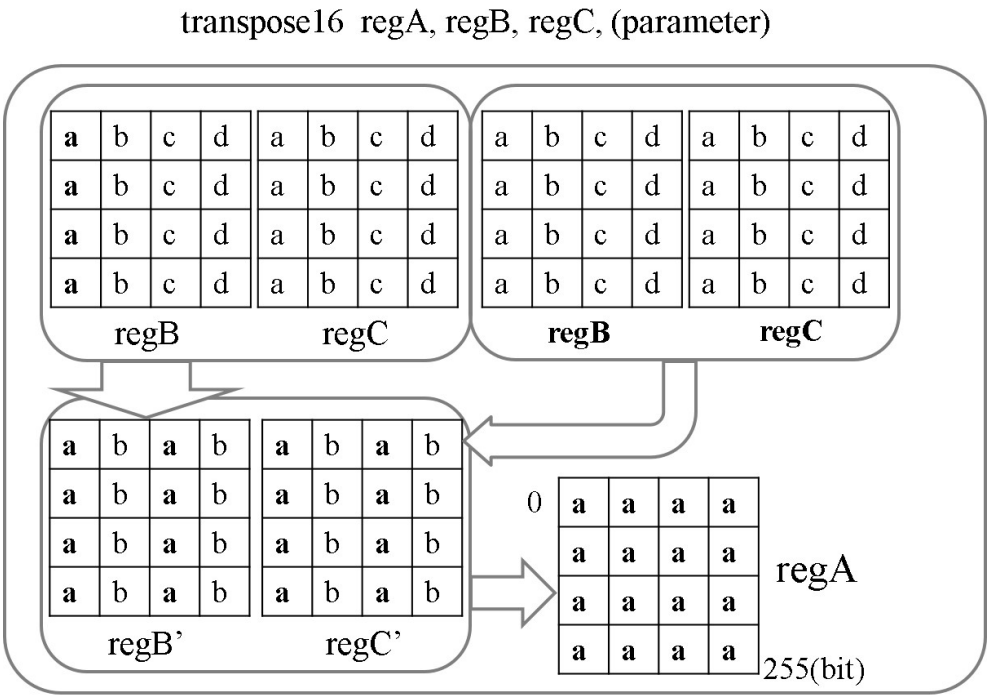

図 5.15: transpose16 命令の実行

たように，交換対称となる列を指定するパラメータとして使い，最終的にレジスタ A には a 列，レジスタ B には b 列，レジスタ C には c 列，レジスタ D には d 列を格納することで，命令の各ソースとディスティネーションの組み合わせによって，一列の縦方向データを抽出し一つのレジスタに格納する．二次元直交変換にこの命令を使用することで，行と列の両方向に対するベクトル演算を，等しい効率で計算することができる．

5.2.2 TRANSPOSE32BIT 命令

この TRANSPOSE32BIT 命令は，32 ビットデータに対する縦方向データ抽出が必要な場合を想定し，実装を行っている．具体的には，タイルデータを持つ二つのレジスタ間で値交換を行い，一つのレジスタに一列の 32 ビットデータを格納する．

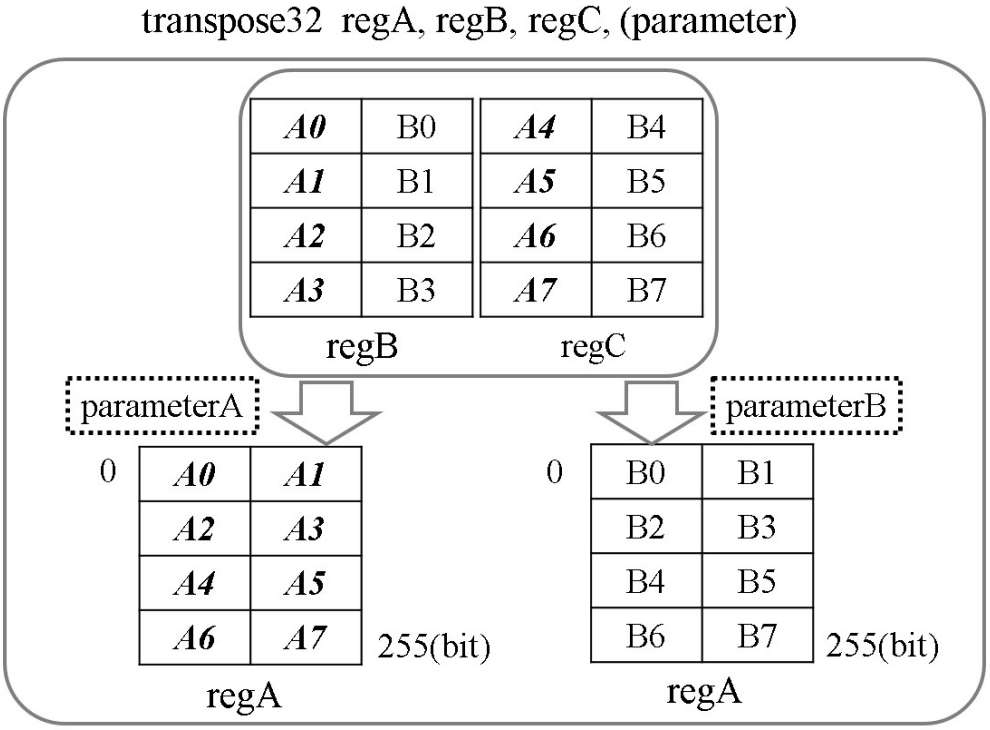

図 5.16: transpose32 命令の実行

命令のソースや，命令で指定したパラメータによって出力する列が異なるなどの細かい仕様は，16 ビットの場合と異なる部分があるが，二つのレジスタ間の入れ替えによって一列のデータを得られるのは同様である．図 5.16 で示した命令を二回使用することで，列データを含むレジスタ間のデータ交換を実現する．この命令によって 32 ビットデータが一列分抽出できる．

5.2.3 TRANSPOSETILE 命令

この命令は，4 × 4 サイズのデータに対する転置を行う．また，上の二つの命令を使って抽出されたデータを，ラインデータとして扱う場合，列内の順序が異なっている場合があるため，この命令を用いて追加の整列を行う．

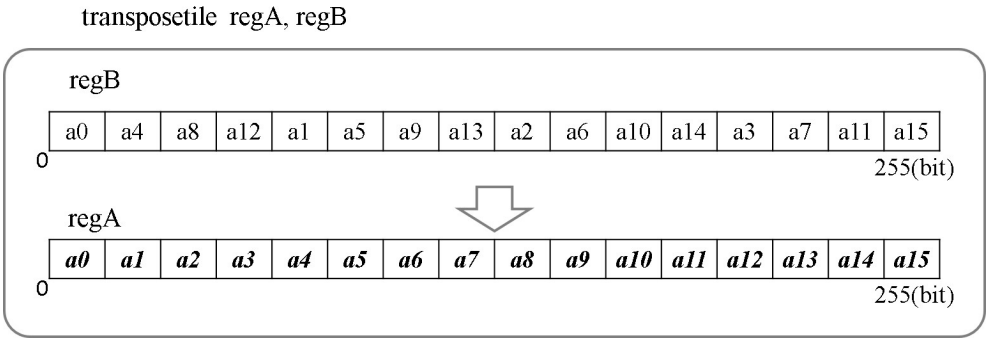

図 5.17: transposetile 命令の実行

この命令の機能は，レジスタ B の値を列方向に整列させたものを，レジスタ A に出力するものである．図 4.12 で示した，タイルアクセスによって得られるデータ配置をライン上に表したものを，図 5.17 のレジスタ B で示している．また，256 ビットのベクトルレジスタに，16 ビットの 4 × 4 データを格納する場合は，タイルデータの転置を実行する命令として扱うこともできる．

5.2.4 SPECIALSHUFFLE 命令

この命令は，図 3.7 で示された，直交変換に要求されるバタフライ演算を，SIMD 演算を用いて計算する場合に要求されるデータシャッフルを，1 命令ステップで実行する．FFT や高速 DCT 変換のデータ入れ替えに対応する全パターンの入れ替えをパラメータで指定し，それに応じたデータシャッフルを実行する．

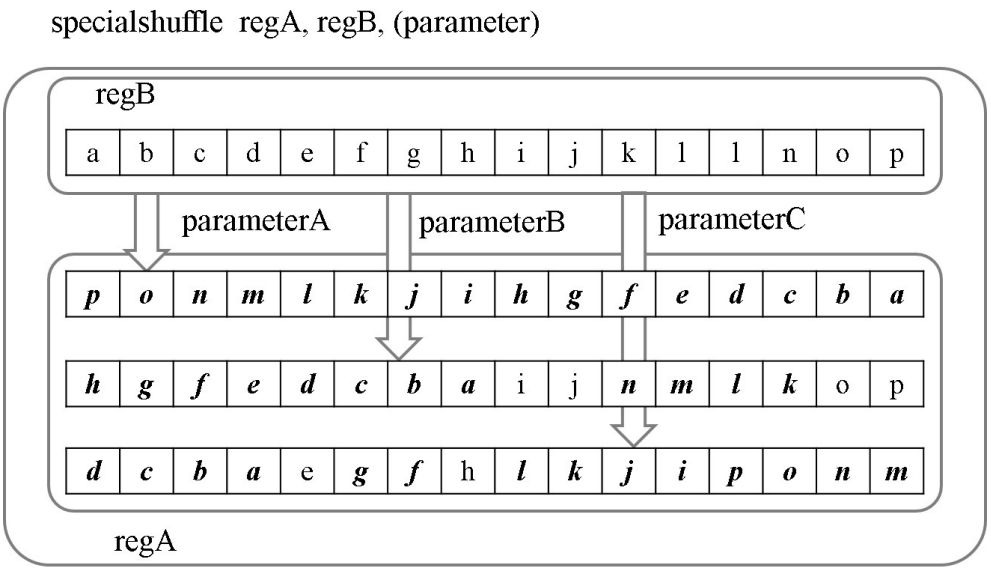

図 5.18: SPECIALSHUFFLE 命令の実行

直交変換の高速化アルゴリズムで示されたバタフライ演算に伴うデータ入れ替えを，パラメータで指定することによって，直交変換処理に最適なデータ入れ替えを実現する．直交変換のベクトル乗算や加算処理の合間に，データ入れ替えが必要になるたびこの命令を実行することで，シャッフル演算ステップの削減に繋げることができる．

6　評価

高速な直交変換を実行する提案手法の優位性を明らかにするため，H.265で定義された変換ブロックで実行する直交変換に必要な実行ステップ数と，各命令の計算機能を実現するために必要な，ベクトル演算器の面積規模を導出した．

ステップ数は以下の各条件を元に算出した．

1．従来の二次元直交変換では，IntelAVX2 は vpshufhw 命令など数種類のシャッフル命令を使い，IntelAVX512 は AVX2 の命令セットで用いたシャッフル命令の代わりに，vpermw 命令を使用する．

2．提案手法の二次元直交変換では，図 5.15 から図 5.18 で示した命令を用いる．

3．二次元直交変換の全体に対するステップ数を求めるために，加算や乗算のステップ数も含めた数値を示す．

4．表に掲載する数値はまず，列方向データ抽出と一次元直交変換，そして二次元直交変換で別々に，所要ステップ数を表す．

5．二次元直交変換は二回の一次元直交変換と，一回の列方向データ抽出が必要になるため，後に挙げる，三つの表に表記するステップ数を足しあわせた数値を使用する．

6．転置処理の効率は，一次元直交変換の変換係数を計算した直後に，レジスタに計算結果が残る場合と，置き換えられる場合で異なる．そのため，列方向のデータ抽出と二次元直交変換のステップ数は，レジスタに残った値を抽出する場合と，データメモリにストアされた値を再ロードして抽出する時の二つを場合分けし，ステップ数を計算する．

表 6.1〜表 6.3 に，H.265/HEVC で定義されている変換サイズに応じた 16 ビットデータの二次元直交変換と，広範な用途で使用される 8 × 8 行列サイズの 32 ビット直交変換を各手法で行った場合の，行列 1 ライン毎の処理ステップの算出結果を示す．

表 6.1: 列方向データ抽出に必要なステップ数

レジスタから列データを抽出する場合			
行列サイズと各手法	AVX2	AVX512	提案手法
16bitDCT(4 × 4)	6	3	1
16bitDCT(8 × 8)	28	22	4
16bitDCT(16 × 16)	112	88	48
16bitDCT(32 × 32)	448	352	192
32bitFFT(8 × 8)	28	25	8
メモリから列データを抽出する場合			
行列サイズと各手法	AVX2	AVX512	提案手法
16bitDCT(4 × 4)	16	13	5
16bitDCT(8 × 8)	44	44	20
16bitDCT(16 × 16)	176	152	112
16bitDCT(32 × 32)	704	608	448
32bitFFT(8 × 8)	44	41	40

表 6.2: 一次元直交変換のステップ数

行列サイズと各手法	AVX2	AVX512	提案手法
16bitDCT(4 × 4)	11	11	11
16bitDCT(8 × 8)	25	25	21
16bitDCT(16 × 16)	57	43	31
16bitDCT(32 × 32)	124	107	79
32bitFFT(8 × 8)	17	17	15

表 6.3: 二次元直交変換のステップ数

レジスタから列データを抽出する場合			
行列サイズと各手法	AVX2	AVX512	提案手法
16bitDCT(4 × 4)	28	25	23
16bitDCT(8 × 8)	78	72	46
16bitDCT(16 × 16)	226	174	110
16bitDCT(32 × 32)	696	566	350
32bitFFT(8 × 8)	62	59	38
メモリから列データを抽出する場合			
行列サイズと各手法	AVX2	AVX512	提案手法
16bitDCT(4 × 4)	38	35	27
16bitDCT(8 × 8)	94	94	62
16bitDCT(16 × 16)	290	238	174
16bitDCT(32 × 32)	952	822	606
32bitFFT(8 × 8)	78	75	70

また，IntelAVX2 拡張命令を使った実行ステップ数を 1 としたときの，IntelAVX512 と提案手法のステップ数を，レジスタデータの縦方向データ抽出とメモリの縦方向データ抽出での二通りに分けて，グラフで示す.

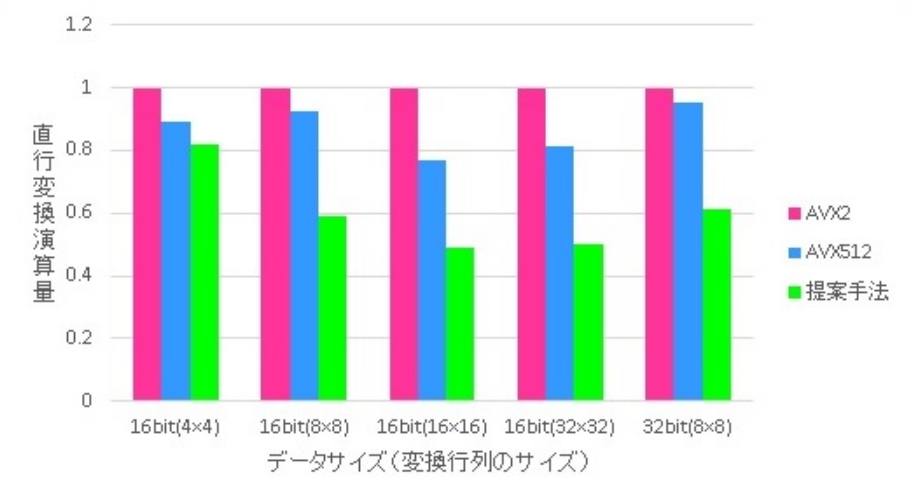

図 6.19: レジスタから列データを抽出する場合の高速化率

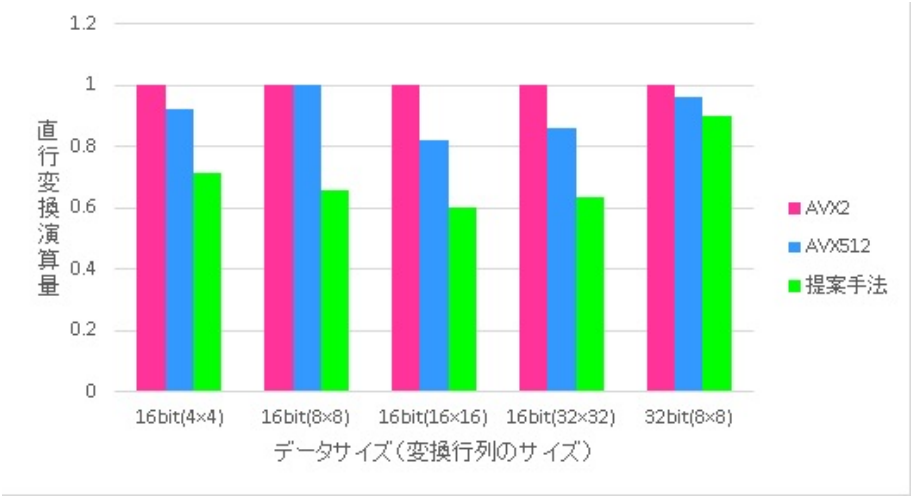

図 6.20: メモリから列データを抽出する場合の高速化率

算出結果から，IntelAVX2 と比べてより新しい命令セットの IntelAVX512 と，提案手法が両方，二次元直交変換の全ての場合でステップ数を削減できていることが分かる．

ステップ数の削減量は，レジスタから列データを抽出する場合，IntelAVX512 は平均で 13 %程度ステップ数を低減できるが，提案手法は平均で 40 %のステップ数を低減でき，最低でも 18%高速 DCT のステップ数を低減できている．またメモリから列データを抽出する場合では，IntelAVX512 は平均で 9 %のステップ数を低減できるが，提案手法は 30 %のステップ数を低減できる．また，横軸の用途別でまとめた特徴としては，16 ビットデータの 8 × 8 サイズ直交変換のような一部のサイズでは，AVX512 の高速化効果が比較的小さく，従来の AVX2 とほとんど変化しない結果になった．

結果の中で特徴的な部分としては，FFT の実行速度に関わる 32 ビット（8 × 8）データでは，前者の場合と後者の場合で提案手法の高速化率が大幅に異なる結果になった．この原因としては，3 章で述べたように，離散フーリエ変換の高速化アルゴリズムである FFT は，DCT の高速化アルゴリズムと比較してデータシャッフルが単純なため，専用的なシャッフル演算機能を用いてもあまり高速化できないためだと推測できる．

そして表 6.4 では，合成ツール synplify を使用して測定した論理合成によって，従来手法の直交変換に主に使用する命令群を実装した計算機能と，提案手法の直交変換に使用する命令群を実装した計算機能に必要なハードウェア規模を，演算器を構成するゲート数として算出した結果を示す．

表 6.4: ハードウェア規模

	演算器面積
IntelAVX2	8504
IntelAVX512	7525
提案手法	6357

表 6.4 の結果は，提案した演算器の面積は他の手法より小さいことを示している．このため，従来より高速な直交変換を，より小さいハードウェア規模で実現できる，提案した計算機能が，最も優れた方法であるといえる．

7　おわりに

本論文では，汎用プロセッサで行われる信号処理のうち，様々な信号処理で使われる DFT と DCT を初めとする二次元直交変換の直交変換を高速化するため，従来のプロセッサの SIMD 演算命令に換わる新たな演算命令を提案し，その機能を持つ演算器の設計を行った．提案する演算機能は，最小限のハードウェアコストで最大限の二次元直交変換の高速化を実現する機能であり，従来の x86 プロセッサをベースとした汎用的なデータパスに，新しい計算機能を追加することで，従来手法と比べて特定の条件で，最大約 50 %高速化できることを示した．

今後の課題は，512 ビットベクトルレジスタを用いた，転置や二次元直交変換のステップ数比較と，スーパースカラー命令発行方式を用いて，さらなる DCT 変換の高速化を検討すること，などが挙げられる．

謝辞

本研究を遂行するに当たり，日頃よりご指導，ご助言をいただきました近藤利夫名誉教授，大野和彦講師，深澤祐樹研究員に感謝いたします．また，様々な面でお世話になったコンピューター・アーキテクチャ研究室の方々と，コンピューター・ソフトウェア研究室の方々に感謝の意を表します．

参考文献

[1] 小久保 潤, 伊藤 聡志, 山田 芳文, “圧縮センシングを導入した MR 高速撮像における GPU 使用による再構成の高速化”, Medical Imageing Technology, Vol. 30, No. 2, Mar. 2012.

[2] Jarno Vanne, et al. “Comparative Rate-Distortion-Complexity Analysis of HEVC and AVC Video Codecs”, IEEE Transactions on Circuits and systems for Video Technology, Vol. 22, No. 12, Dec. 2012.

[3] ITU-T Recommendation H.265.2, “Reference software for ITU-T H.265 high efficiency video coding”, Apr. 2015.

[4] Intel Corporation, “インテル 64 アーキテクチャーおよび IA-32 アーキテクチャー最適化リファレンスマニュアル”, 2011 年 4 月.

[5] Richard Hubbard, “Using Intel Advanced Vector Extensions to Implement an Inverse Discrete Cosine Transform”, Intel Corporation, Sep. 2010.

[6] 渡邊 敬太, “高効率動き検出に対応する SIMD 併用型汎用データパス構成法の研究”, 三重大学, 2014 年.

[7] B.K.Wang, et al. ”Tile/line access cache memory based on a multi-level Z-order tiling data layout”, Concurrency Computation Practice and Experience, [Online]: https://doi.org/10.1002/cpe.4375, Dec. 2017.

[8] Jong-Sik Park, Woo-Jin Nam, Seung-Mok Han, Seongsoo Lee, “2-d Large Inverse Transform (16x16, 32x32) for HEVC (High Efficiency Video Coding)“, Journal of Semiconductor Technology and Science, Vol.12, No.2, jun. 2012.

[9] Wen-Hsiung Chen, et al. “A Fast Computational Algorithm for the Discrete Cosine Transform”, IEEE Transactions on Communications, Vol.Com-25, No.9, pp.1004–1009, Sep. 1977.

[10] 大久保, 鈴木, 高村, 中篠, H.265/HEVC 教科書, 株式会社インプレスジャパン, 2013 年 10 月.

[11] Intel Corporation, “Intel 64 and IA-32 Architectures Optimization Reference Manual”, Apr. 2018.

[12] Intel Corporation, “Intel Xeon Scalable Processor (Instruction Throughput and Latency)”, Aug. 2017.

[13] x265HEVC Encoder/H.265 VideoCodec, [Online]: http://x265.org/.

[14] 中島 耕太, 佐藤 充, 後藤 正徳, 住元 真司, 久門耕一, 石川 裕,“配列転置データ転送を高速化する 10Gb Ethernet インタフェースカードの設計”, 情報処理学会論文誌 コンピューティングシステム, Vol47, No.SIG 12 (ACS 15), 2006 年 9 月.