

修士論文

題目

SIMD型並列処理に適した
タイル/ライン両対応キャッシュの
提案

指導教員

近藤 利夫 教授

平成 25 年度

三重大学大学院 工学研究科 情報工学専攻
計算機アーキテクチャ研究室

猪俣 匠 (412M503)

内容梗概

近年，データパスの高度化によりプロセッサ自体の演算性能は大きく向上しているにも関わらず，メモリアクセスがボトルネックとなり，本来の性能を引き出せないアプリケーションが少なくない．特に動画像処理などのマルチメディアアプリケーションや行列計算に対して，複数データに同じ演算を行う SIMD 拡張命令の高並列演算能力を十分に活かすことは出来ていない．2次元データ処理においては，キャッシュラインサイズを拡大しても，2次元の空間的局所性のおかげで，キャッシュラインコンフリクトを低減出来ないばかりか，キャパシティーミス増大の要因になってしまうからである．このように，従来のキャッシュメモリをそのままの構成で用いるのでは，SIMD 演算器の性能を引き出すのは困難である．また行列計算においても，キャッシュラインに行方向のデータの並びが格納されることから，行列の転置や列方向の要素データの並び替えなどが，高速化のボトルネックとなっている．

本論文では，このような状況を大幅に改善するタイル/ライン両アクセス対応の新構成のキャッシュメモリを提案している．提案キャッシュメモリは，データアライメント制約の解除機構と，タイルデータ割り当てによる2次元ブロックデータアクセス機構を併せ持つ構成を採っている．このような構成を採ることで，不必要なデータをキャッシュメモリに格納する割合を減らし，キャパシティーミスを低減するのに加え，2次元の局所性を有するデータアクセスにおけるキャッシュラインコンフリクトミスも低減している．

この提案キャッシュの性能評価のために，動画像符号化ソフトウェア内に機能シミュレータを組み込み，従来キャッシュと比較して評価を行った．その結果，データ転送に要するサイクル数を約 30%低減することが出来た．

Abstract

In recent years, since operation performance of the processor greatly increased due to the various improvements of the data path. Memory access rate frequently becomes bottleneck in high-speed computation. For example, it is not possible to take full highly parallel computing performance of SIMD operation at matrix calculation and multimedia application such as video encoding. In the two-dimensional data access, the more cache line size increases, the more cache line conflict miss and capacity miss increase. Therefore it is difficult to bring out the computing performance of SIMD operation by using just a conventional cache memory. In addition, transpose of the matrix and rearrangement the raster data in the column direction becomes bottleneck to accelerate a matrix calculation. Because the conventional cache stores data of the raster format in the cache line.

In this paper, we propose a new cache memory with both tile/line unit accessibility. Proposed cache memory is compatible with non-aligned data access and two-dimensional data access. It is possible to reduce the proportion that stores unnecessary data in the cache memory. Therefore, proposed cache memory reduce the cache capacity miss and cache line conflict miss to the two-dimensional data access.

For performance evaluation, we have added functional simulator of the proposed cache memory to software video encoder and evaluated its performance in the motion estimation. As a result, data loading cycle decreased compared with normal cache.

目次

1	まえがき	1
2	従来アクセス方法の問題点	3
2.1	データアライメント制約	3
2.2	キャッシュラインコンフリクトミス	4
2.2.1	2次元ブロックデータアクセス	4
2.2.2	動き検出処理	5
2.2.3	インデックスコンフリクト	6
2.3	転置処理	7
2.3.1	DCT 処理	7
2.3.2	行列計算処理	9
3	関連研究	10
3.1	Two-bank interleaved cache	10
3.2	キャッシュブロッキング手法	11
3.3	Block offset mapping	11
3.4	Split-index mapping	12
4	提案キャッシュメモリ	14
4.1	タイルデータ割り当て	14
4.2	データアライメント制約解除機構	16
4.2.1	スキュードアレイによるデータ格納	16
4.2.2	アドレス変換機構	18
4.2.3	タイル/ラインデータの転送処理	18
4.3	2次元データ転送と転置処理の高速化	21
5	評価	24
5.1	評価環境構築	24
5.2	性能評価	25
5.3	ハードウェア規模評価	28
5.3.1	タグメモリの容量削減	28
5.3.2	結果	29
6	あとがき	31

謝辞	32
参考文献	32
A 実験手順	34
A.1 シミュレーションの実装手順	34

目 次

2.1	データアライメント制約	3
2.2	ブロックデータアクセス	5
2.3	可変ブロックサイズ	6
2.4	インデックス競合	7
2.5	4x4 の行列転置	9
3.6	Two-bank interleaved cache	10
3.7	行列積の (a) ブロッキングなし, (b) ブロッキング適用時の アクセスパターン	12
3.8	Block offset mapping	12
3.9	Split-index mapping	13
4.10	タイルデータ割り当て時のブロックデータアクセス	15
4.11	データアライメント制約解除機構	16
4.12	各メモリバンクへのデータ格納	17
4.13	スキュードアレイ形式でのデータ格納	17
4.14	ビット入れ替えによるアドレス変換	18
4.15	アドレス変換部詳細	19
4.16	キャッシュヒット/ミス判定	19
4.17	ラインデータ読み出し	20
4.18	タイルデータ読み出し	21
4.19	シャッフル命令を用いた転置処理の高速化	22
5.20	HorseRace 画像での評価	26
5.21	Bronze with credits 画像での評価	26
5.22	連想度別評価	28
5.23	タグメモリの容量削減	29

表 目 次

2.1	x264 内部の DCT スループットカウント	8
4.2	アクセス回数削減率	15
4.3	転置処理の所要サイクル数	22
5.4	メモリ構成	24
5.5	符号化条件	24
5.6	キャッシュのヒット率	25
5.7	キャッシュライン境界交差率と平均アクセスサイクル	27
5.8	ハードウェア規模評価	30

1 まえがき

近年、プロセッサの性能向上によりアプリケーションの実効性能が大幅に向上している。動画画像処理などのマルチメディアアプリケーションや科学技術計算などで使用される行列計算については、複数データに対して同じ演算を行う SIMD(Single Instruction stream Multiple Data stream) 演算を用いた並列処理が広く使用されている。このような並列処理の性能向上に向けて x86 プロセッサには AVX(Advanced Vector eXtensions) と呼ばれる SIMD 拡張命令セットが搭載されている。しかし、SIMD データバスへのデータ供給がボトルネックとなり、本来の性能を引き出せないことも少なくない。

例えば、動画画像処理や行列計算処理においては、頻繁に発生する 2 次元のデータアクセスに対し、従来のキャッシュメモリではうまく機能しない。何故なら、従来のキャッシュメモリは 1 次元データのアクセスにおいてはデータ局所性が高くなるように割り当て可能であるが、2 次元データのアクセスにおいてはデータ局所性が発揮されず、キャッシュラインサイズを拡大してもキャッシュラインコンフリクトを低減できないばかりか、キャパシティーミス増大の要因となってしまうからである。このように、従来構成のままでは、キャッシュメモリがボトルネックとなり、SIMD 演算器の性能が引き出せない。

本論文では、このような状況の大幅な改善を目指し、データアライメント制約の解除機構と、タイルデータ割り当てによる 2 次元ブロックデータアクセス機能を併せ持つ構成を採ることで、タイル/ライン両アクセスに対応する新構成のキャッシュメモリを提案する。このタイル/ライン、すなわちラスタ走査順の 1 次元画素配列で構成されるラインデータと、2 次元スモールタイルで構成されるタイルデータの両方のデータアクセスを、データメモリのマルチバンク化とスキュードアレイによる各メモリバンクへの格納を組み合わせることにより実現する。キャッシュメモリに対するタイルデータ割り当て手法としては、キャッシュラインと同サイズのスモールタイルとキャッシュメモリと同サイズのラージタイルを組み合わせた階層的なタイル構成を採る。このような構成を採ることで、不必要なデータをキャッシュメモリに格納する割合を減らし、キャパシティーミスが低減するのに加え、2 次元の局所性を有するデータアクセスにおけるキャッシュラインコンフリクトミスの低減も図る。さらに、圧縮符号化で使用される DCT 処理、科学技術計算において広く用いられる FFT や行列計算等で、共通して処理上のネックとなっている 2 次元配列データの

転置処理の高速化手法を提案する．また，性能評価のために，動画像符号化ソフトウェア内にキャッシュメモリの機能シミュレータを組み込み，従来キャッシュと提案キャッシュに比べ，動き検出処理におけるデータ転送時間，DCT 係数行列の転置処理時間等がどれだけ低減されるかを明らかにする．

2 従来アクセス方法の問題点

2.1 データアライメント制約

キャッシュメモリへのデータ格納はその性質上、キャッシュライン単位でメモリから転送される。キャッシュメモリから SIMD 演算器にデータ供給するとき、キャッシュメモリのキャッシュライン単位でのアクセス制約(データアライメント制約)がデータ転送のボトルネックとなっている。

SIMD 演算器の性能をフルに活かすためには、演算に必要とするデータを一つのレジスタに格納する必要があるが、必要とするデータがキャッシュメモリ内の複数のキャッシュラインにまたがって格納されていることがある。図 2.1 にその例を示す。この場合、従来キャッシュではハードウェア的に 1 サイクルには 1 つのキャッシュラインのアクセスに限定されているため、関係するキャッシュラインの数だけデータ供給にかかるサイクル数が増えることになる。さらに、読み出したキャッシュラインから必要データ部分を抜き出すための処理も必要となってしまう。そのデータ抜きだし処理についても、SIMD 演算時のオーバーヘッドとなる。動画処理などのマルチメディアアプリケーションではこのような複数のキャッシュラインにまたがったデータアクセスが頻繁に発生し、データ供給時のボトルネックとなる [1]。

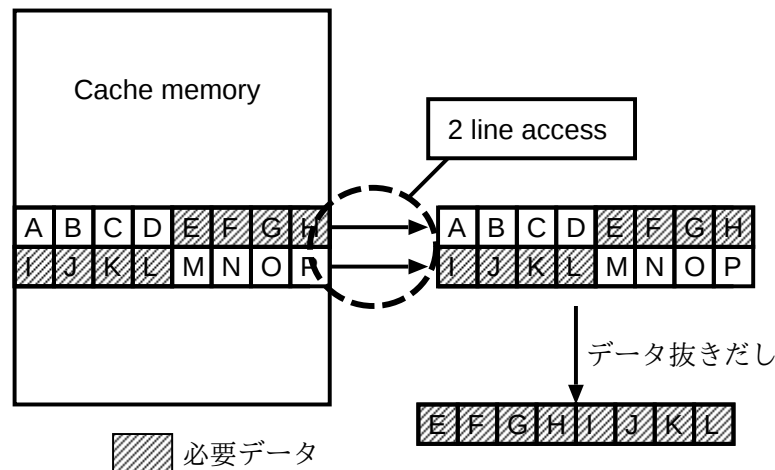


図 2.1: データアライメント制約

2.2 キャッシュラインコンフリクトミス

キャッシュメモリに格納できるデータ量には制限があり，主記憶内の全てのデータを格納できるわけではない．要求するデータがキャッシュメモリ内に存在しない場合，必要データを含むキャッシュラインをさらに下層のメモリから取り出す必要があり，実効性能が低下してしまう．プログラムの実効性能を向上させるためには，キャッシュのミス回数を削減することが重要である．キャッシュミスは，初めてデータにアクセスする際に起こる初期参照ミス，処理するデータサイズがキャッシュサイズを超えることによって起こるキャパシティーミス，キャッシュの連想度が足りないためにインデックス競合が発生することによって起こるコンフリクトミスの3種類に分類される．

初期参照ミスは発生回数が少なく，初期参照時に必ず発生するミスのため，削減するのは難しい．キャパシティーミスは，キャッシュメモリ容量やキャッシュラインサイズを増やすことにより削減可能である．以上から，特にダイレクトマップなどの連想度が低いキャッシュメモリにおいては，コンフリクトミスの削減が重要であることが知られている．

2.2.1 2次元ブロックデータアクセス

従来キャッシュメモリでは画像データはラスタ走査順にメモリに格納されているため，キャッシュメモリに格納されるデータは横方向に連続する一次元的なデータである．しかし，動画像処理では2次元ブロックデータ単位の処理が多く，従来キャッシュのようなラスタ走査順の格納方法では2次元データの空間的局所性が無いため，キャッシュラインコンフリクトミス増大の要因となってしまう．

図2.2にキャッシュラインサイズ N_{byte} のキャッシュメモリにおける2次元ブロックデータアクセスの例を示す．キャッシュラインのサイズがメモリの最小アクセスサイズとなるため，使用しないデータの転送が不可避となる．さらに，ブロックデータの縦幅の分だけメモリアccessが必要のため，メモリアccess回数も増えてしまう．図2.2(a)のようにブロックデータの1ライン分が1つのキャッシュラインに格納される場合は，アクセス回数と不要なデータ転送はある程度抑えられるが，図2.2(b)のように1ラインがキャッシュライン境界を跨いでいる場合には，アクセス回数と不要なデータ転送が大幅に増えてしまう．さらに，前述のデータアライメント制約による性能低下も発生してしまう．

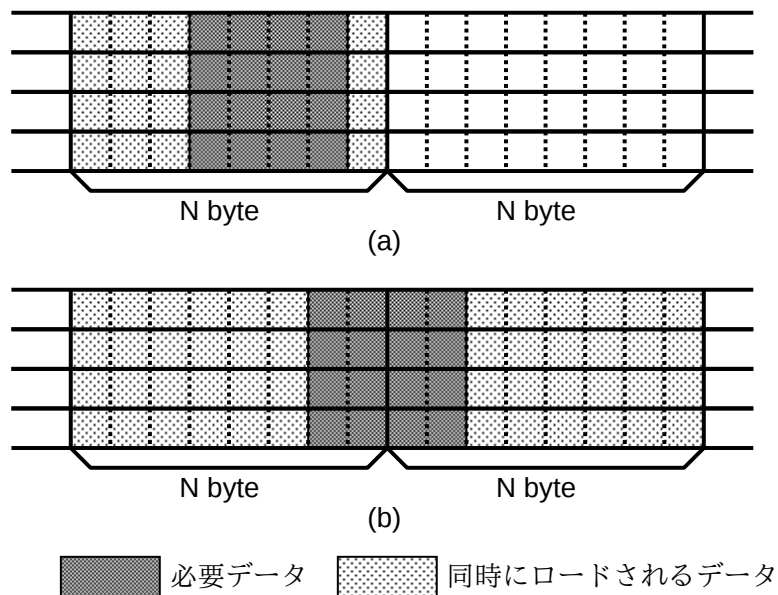


図 2.2: ブロックデータアクセス

2.2.2 動き検出処理

動画画像符号化処理では、圧縮率を高めるために動き検出処理が行われる。動き検出は、マクロブロック単位でブロックマッチングを行い、参照画像の中で符号化対象ブロックと最も類似している箇所を検出し、動きベクトルを算出する。この算出した動きベクトルを用いて、動きの差分のみを圧縮することにより、冗長なデータを減らすことができ、画素値をそのまま符号化するのに比べて大幅な圧縮率向上が見込める。ブロックマッチング処理においては、参照画像内で2次元ブロック単位でのデータアクセスが必要となり、2次元データ空間にたいして参照の局所性を持っている。そのため、従来のラスタ走査順の格納方法ではアクセス効率が悪くなってしまう。また、その際のデータアクセスの大半は、参照画像内を少しずつ移動しながら探索を行うことから、複数のキャッシュラインにまたがったデータアクセスとなる確率が高く、前述のデータアライメント制約による性能低下も発生してしまう。

動き検出処理は動画画像符号化処理中で最も処理量が多く、高速化のボトルネックとなっている [2]。現在主流である符号化圧縮規格の H.264/AVC では、動き検出処理において可変ブロックサイズが使用されている。この可変ブロックサイズは全部で7種類あり、16x16, 16x8, 8x16, 8x8, 8x4,

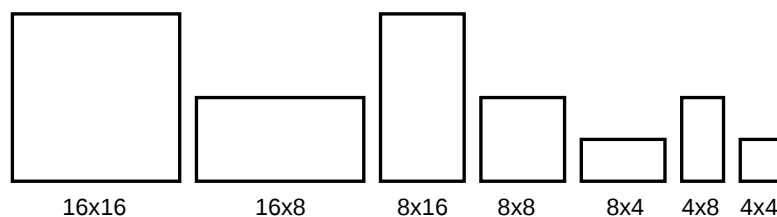


図 2.3: 可変ブロックサイズ

4x8, 4x4 のブロックサイズとなる。7 種類の可変ブロックサイズを図 2.3 に示す。キャッシュメモリの 1 ラインのサイズは 32-128byte であり、最大でも 16x16 のブロックに対しては、不要なデータを余分に読み込む割合が高く、非効率となる。キャッシュラインサイズが 32byte で 16x16 画素のマクロブロックを読み出そうとすると、1 つのキャッシュラインに必要な画素ブロック構成ラインが収まる最良の場合でも 16x16 画素分の不必要なデータ転送が発生し、2 つのキャッシュラインにまたがってしまった場合には 48x16 画素もの不必要なデータ転送が発生してしまう。

2.2.3 インデックスコンフリクト

従来のキャッシュメモリでは、横方向に連続するデータに対しては別々のインデックスが割り当てられるため、横方向に連続するデータがキャッシュメモリの同一箇所に割り当てられることはない。しかし縦方向に連続するデータに対しては、データが連続していないため、別のインデックスが割り当てられている保証はない。画像の横幅がキャッシュメモリ容量の整数倍に一致する最悪の場合を図 2.4 に示す。この場合では、16x16 のブロックデータアクセス時にインデックスコンフリクトによるキャッシュミスが 15 回起こってしまう。さらに、キャッシュメモリに格納されるデータは最後にアクセスした 1 ラインのみとなってしまうため、データの再利用性が全く活かせない。

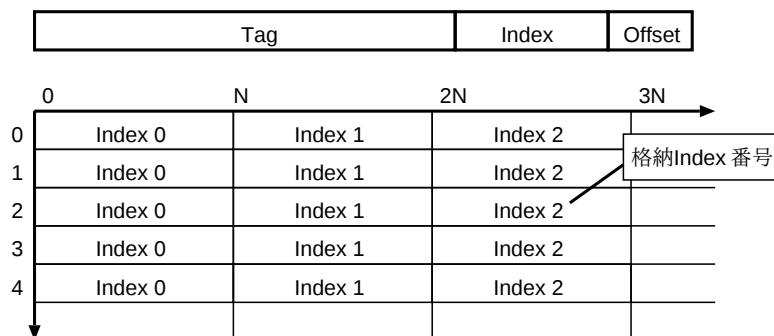


図 2.4: インデックス競合

2.3 転置処理

2.3.1 DCT 処理

動画画像符号化処理では圧縮率を高めるために、動き検出処理の他に 2 次元離散コサイン変換 (DCT) 処理が行われる。DCT とは画素を別の情報である周波数に変換するものである。一般に、人間の眼は低周波成分に対して敏感であり、高周波成分に対しては鈍感であるという特性を持っている。その特性を利用し、高周波成分を削除してしまうことで、画質の劣化を感じさせることなく圧縮率の向上が行える。

DCT 処理は動画画像符号化処理中で動き検出処理の次に処理量が多く、動画画像符号化処理だけでなく、画像圧縮や音声圧縮などマルチメディア処理の大半で使用されている [3]。DCT 処理の高速化のために、オープンソースで使用される H.264 符号化プログラムである VideoLANx264 を分析して、ボトルネックとなっている箇所を分析した。VideoLANx264 は MMX や SSE, AVX などの x86 プロセッサの各 SIMD 拡張命令に対応している [4]。x264 内部での DCT 処理スループットを見積もった結果を表 2.1 に示す。x264 内部では、4x4, 8x8, 16x16 の 3 種類のブロックサイズで DCT 処理を行う関数がある。DCT 関数の処理手順を以下に示す。

1. 処理対象ブロックのロード (Memory Access)
2. 横方向への DCT 処理 (1-D DCT)
3. 処理対象ブロックの転置処理 (Transpose)
4. 横方向への DCT 処理 (1-D DCT)

表 2.1: x264 内部の DCT スループットカウント

Block size	Memory Access	1-D DCT	Transpose	1-D DCT	Others	Total
DCT4x4residual(sub<block_size>.dctfunction)						
4x4	10.5	9	13	9	8	49.5
8x8	8	9	18	9	28.75	72.75
16x16	48	36	72	36	53.5	245.5

5. 処理対象ブロックのストア (Memory Access)

表中の Others は各処理に分類されない副処理を示す．分析の結果，転置処理が DCT 処理全体の 25-30% と多く，ボトルネックとなっていると考えられる．

2次元 DCT 処理は，行方向の 1次元 DCT 処理と列方向の 1次元 DCT 処理を行うことで実現しており，2回の DCT 処理の間でデータの並びを調整するために転置処理が使用されている．転置処理については専用の命令が存在せず，unpack 命令により，データの並びを少しずつ入れ替えることによって実現している．x264 における転置処理の概要を図 2.5 に示す．4画素 1ラインのデータを 4つのレジスタに格納し，その内の 2つのレジスタに対して上位画素の並び替えと下位画素の並び替えの 2命令を計 4回繰り返すことで転置処理を行っている．4x4 の行列転置のために 13 命令必要であり，その内の 8 命令が unpack 命令で，残りがレジスタの中身を入れ替える命令である．前述の通り画素データはラスタ走査順で格納されているため，画素ブロックをライン毎に分割して各レジスタに格納しなければならず，このような非効率な処理になったのだと考えられる．

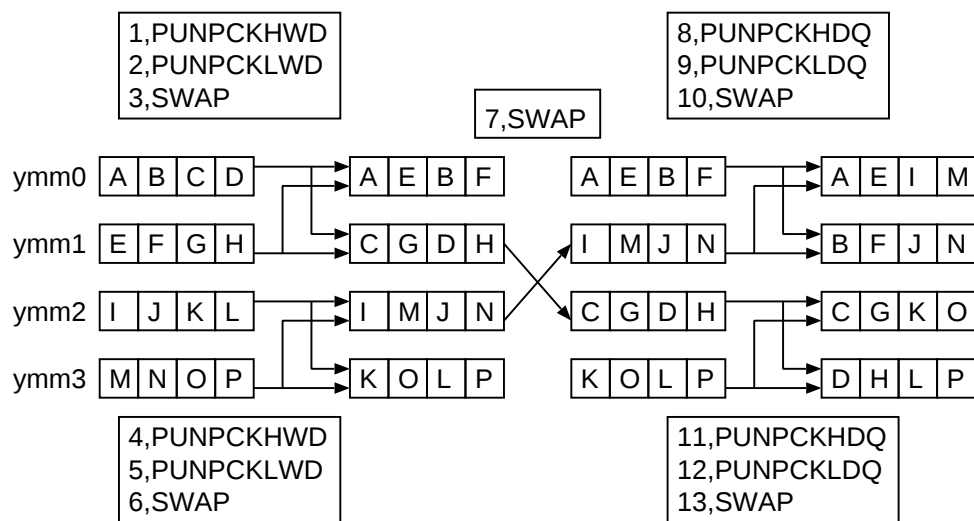


図 2.5: 4x4 の行列転置

2.3.2 行列計算処理

科学技術計算で広く用いられている FFT や行列計算処理においても、全体の副処理として行列転置が用いられている。FFT は離散フーリエ変換を高速に処理するためのアルゴリズムであり、並列処理による高速化アルゴリズムでは、処理の一部に転置処理が複数回使用されており、ボトルネックとなっている [5]。また行列計算処理においては、多用される基本処理の一つであり、高速化が不可欠となっている。行列転置処理に関しては x86 プロセッサの SIMD 命令によって高速化が検討されているが、キャッシュメモリのデータアライメントやラスタデータでの格納などの制約により、前節に示したような複数命令を用いる以上の高速化は実現できない。

3 関連研究

2章で述べたデータアライメント制約や、キャッシュラインコンフリクトミスのようなキャッシュメモリの特性に関連する問題を解決するために、以下に示すようにさまざまな研究が行われている。

3.1 Two-bank interleaved cache

Two-bank interleaved cache ではデータアライメント制約によるオーバーヘッドを、ハードウェアのサポートにより改善する [6]。具体的にはキャッシュメモリを二つのメモリバンクに分割し、偶数アドレスで割り当てられるデータを偶数バンクに、奇数アドレスで割り当てられるデータを奇数バンクに格納することで、必要とするデータが二つのキャッシュラインに分割して格納されている場合でも1サイクルでアクセス可能となる(図3.6)。さらに、二つのキャッシュラインから読み込んだデータの結合と、必要部分の抜き出し処理をデータ供給時にハードウェアを用いて行うことで、データアライメント制約を気にせず、実行可能となる。しかし、この手法では2次元ブロックデータアクセスにおける空間的局所性を活かすことができないため、キャッシュメモリのヒット率は向上しない。

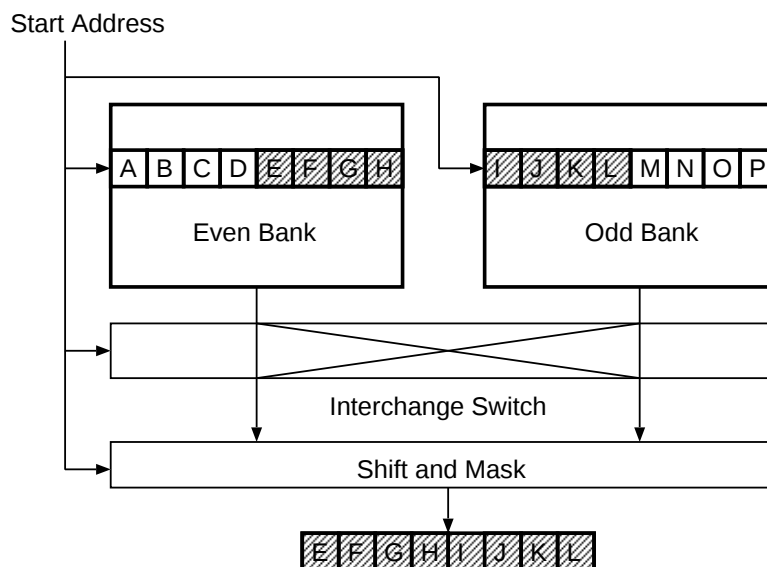


図 3.6: Two-bank interleaved cache

3.2 キャッシュブロッキング手法

ブロッキングは、多くのアプリケーションにおける一般的な最適化手法であり、メモリ帯域幅のボトルネックを回避するのに使用される。具体的には、大規模な画像や行列の全てを処理するのではなく、複数個に分割したブロック単位で処理を行う(図 3.7)。プログラム中では、大規模ループを複数の小規模ループに分割することで参照の局所性を高め、キャッシュや主記憶の利用効率を高めたり、転送のオーバーヘッドを低減したりすることができる。処理ブロックのサイズについては、キャッシュメモリに収まるようにデータ構造をブロッキングするのが望ましい。このブロッキング手法により、2次元の空間的局所性を活かすことが可能となり、画像処理や大規模な行列計算処理においてメモリ効率を大きく向上させることが可能となる。処理するブロックサイズはキャッシュサイズに依存するため、プロセッサ毎に異なる最適化が必要となるものの、ハードウェアに変更を加えることなくキャッシュメモリの効率化が可能であり、ソフト開発においては有用な手法として古くから研究されている [7]。しかし、動き検出処理のような画像データ内をランダムにアクセスするような場合には、複数の処理ブロックをまたがったデータアクセスが頻発してしまい、必要データのアドレス計算が困難であることから、ブロッキング手法は適用できない。

3.3 Block offset mapping

Block offset mapping とは縦横両方向の 2 次元空間に対して空間的局所性を持たせることができるデータ割り当て方法である [8]。具体的には、キャッシュメモリに対するアドレスのオフセット割り当てを変更し、従来のようなラスタデータ単位での格納ではなく、特定のブロックを一つのキャッシュラインに格納することで 2 次元データアクセスを効率的に行うことが可能となる(図 3.8)。この手法を用いることで、2 次元ブロックデータアクセス時の左右方向に対する冗長なデータアクセスを減らすことができ、不必要なデータをキャッシュラインに格納する割合を減らせる。しかし、この手法では従来からのラスタ走査順を前提としたプログラムの大幅な修正が必要になる上に、必要なデータが複数のキャッシュラインに存在する確率が高くなるため、データアライメント制約による性能低下が発生してしまう。

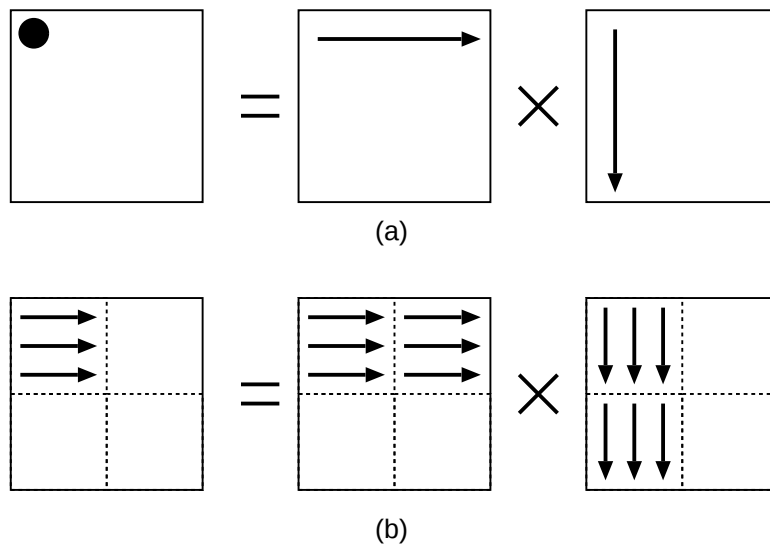


図 3.7: 行列積の (a) ブロッキングなし, (b) ブロッキング適用時のアクセスパターン

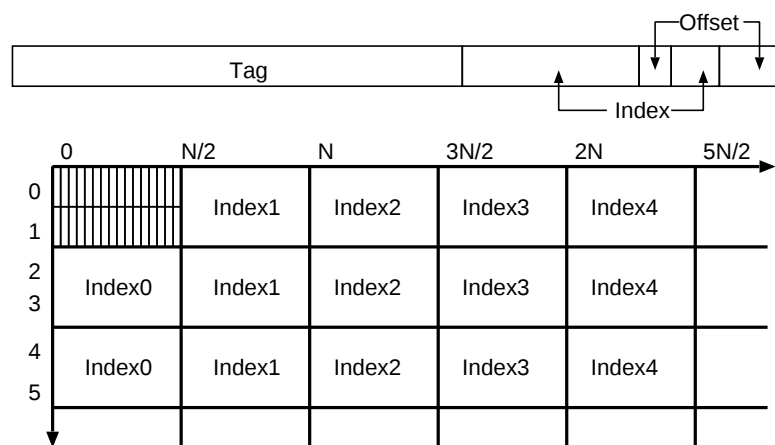


図 3.8: Block offset mapping

3.4 Split-index mapping

2.2.3 節で述べたインデックスコンフリクトによるキャッシュミスを低減するために, 横方向に連続するデータだけでなく, 縦方向に連続するデータに対してもインデックス競合が発生しない領域を拡張することが可

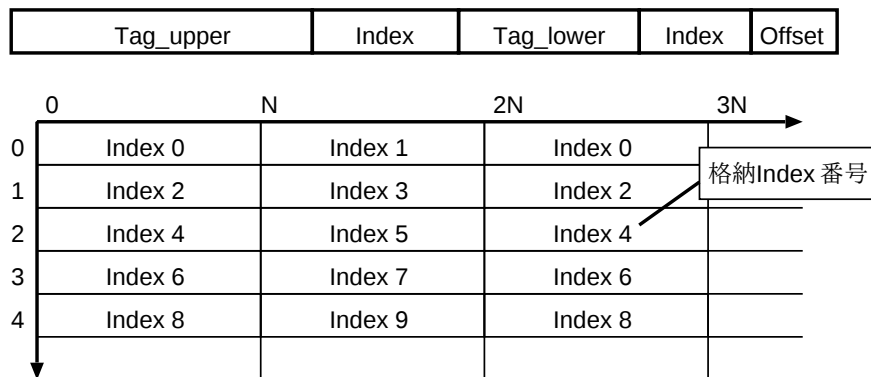


図 3.9: Split-index mapping

能となる Split-index mapping が提案されている [9] . Split-index mapping ではアドレス上のインデックスの割り当てを , 従来の割り当て方法ではなく , インデックス領域を二つに分割することにより , インデックス競合が起きない領域を 2 次元空間に拡張している (図 3.9) . この手法を用いることにより , 縦方向アクセス時のインデックス競合によるラインコンフリクトミスを低減し , キャッシュ性能を向上させることが可能となる . しかし , 汎用プロセッサに適用するには , 2 次元配列の幅に応じて Tag-lower のビット幅を自動的に調整する仕組みが必要になる .

4 提案キャッシュメモリ

3章で説明したように従来のブロッキング手法は，ラスト走査順を前提としたプログラミング手法の大幅な見直しを要求する上に，アドレス計算によるオーバーヘッドを生じる問題もある．また，マルチメディア処理や行列処理の高速化に有効な SSE や AVX に代表される SIMD 拡張命令セットのスループットに対し，キャッシュメモリからのデータ供給性能が追いついていないため，SIMD 命令の性能は制限されてしまう．

これらの問題を解決するために，2次元データアクセスに適した新構成のキャッシュメモリを提案する．この提案キャッシュメモリは，データアライメント制約の解除機構と，タイルデータ割り当てによる2次元ブロックデータアクセス機能を併せ持つ構成を採っている．さらに，ラスト走査順の1次元画素配列で構成されるラインデータと，2次元スモールタイルで構成されるタイルデータの両方のデータ転送に対応している．

4.1 タイルデータ割り当て

タイルデータ割り当ては2次元ブロックデータアクセスに適したデータ割り当て手法である．具体的には画像内全体を画素単位でラスト順に格納するラスト形式データでなく，矩形タイル状に区切り，タイル内は画素単位のラスト順，タイル間はタイル単位のラスト順でそれぞれ格納する．図4.10に8x4サイズでのタイルデータ割り当ての例を示す．16x16の2次元ブロックデータを読み出す際には，図4.10(a)に示す最良の場合は冗長なデータ転送を完全に抑えることができ，図4.10(b)に示す最悪の場合でも，冗長なデータ転送を224byteに抑えることができる．また，冗長なデータ転送を減らすことで，2次元ブロックデータアクセス時のメモリアクセス回数も減らすことができる．さらに，不必要なデータをキャッシュメモリに格納する割合を減らすことができるため，小容量のキャッシュメモリにおいても容量性ミスが減らし，ヒット率の向上が見込める．

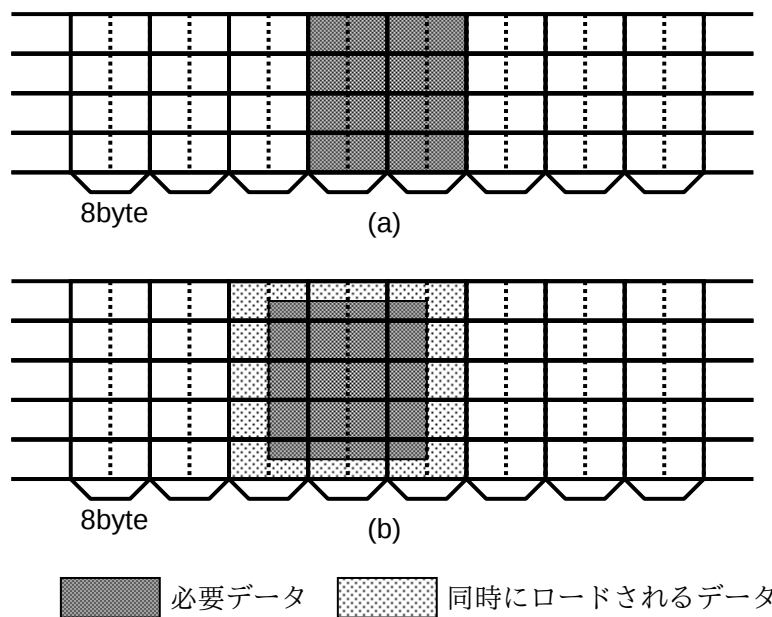


図 4.10: タイルデータ割り当て時のブロックデータアクセス

表 4.2 に従来のラスタデータとタイルデータ割り当てのマクロブロックサイズ別のアクセス回数比較を示す．タイルデータ割り当てを用いることで，従来のラスタデータ割り当てに比べ，アクセス回数を平均 44%削減することが可能となる．

表 4.2: アクセス回数削減率

ブロックサイズ	アクセス回数						
	16x16	16x8	8x16	8x8	8x4	4x8	4x4
ラスタデータアクセス (32x1)	23.5	11.75	19.5	9.75	4.88	8.75	4.38
タイルデータ割り当て (8x4)	13.66	7.91	8.91	5.16	3.28	3.78	2.41
メモリアクセス回数 削減率	41.8%	32.7%	54.3%	47.1%	32.7%	56.8%	44.9%

4.2 データアライメント制約解除機構

関連研究の Two-bank interleaved cache と同様に，キャッシュメモリを複数のメモリバンクで構成することにより，必要データを含むキャッシュラインを競合なしで読み込み可能となる．図 4.11 に提案キャッシュのデータアライメント制約解除機構を示す． 8×4 バイトのタイルデータを，一つのキャッシュラインが 8 バイトで構成される 4 つのメモリバンクに格納する．アドレスからインデックスの抽出を行い，そのインデックスを用いて各メモリバンクにデータアクセスを行う．1 つのメモリバンクから 8 バイトずつデータを転送し，合計 32 バイト分のデータに対し，シフトと抽出を行うことで，必要データの抜き出しを行う．しかし，タイルデータ割り当てとデータアライメント制約解除機構を両立させるためには解決しなければならない問題がある．次節以降にその問題点と解決方法を示す．

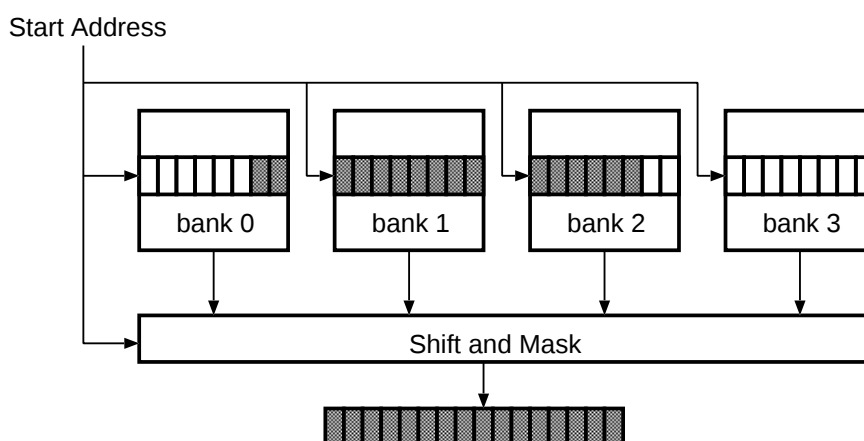


図 4.11: データアライメント制約解除機構

4.2.1 スキュードアレイによるデータ格納

複数バンクにタイルデータを分割して格納するだけでは，ラインデータのアクセス時に 1 つのバンクにアクセスが集中し，並列アクセスできない．図 4.12 に通常の格納形式での各メモリバンクへのデータ格納を示す．タイルデータのアクセス時には，バンク競合を起こさずにデータアクセス可能だが，ラインデータのアクセス時には，要求データが全て同

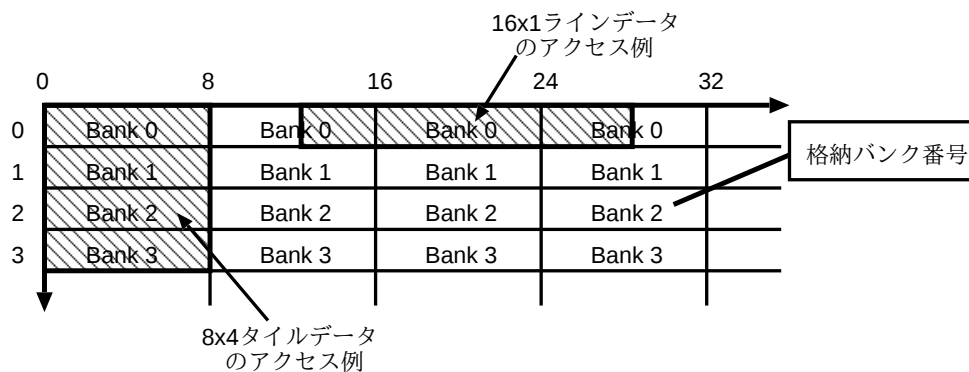


図 4.12: 各メモリバンクへのデータ格納

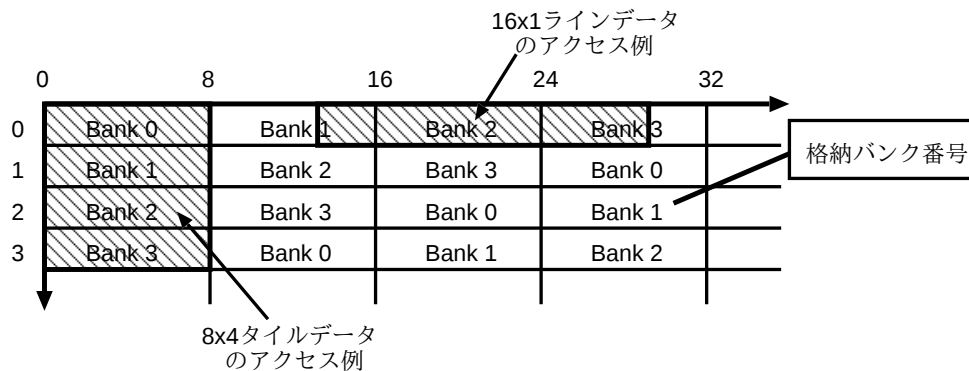


図 4.13: スキュードアレイ形式でのデータ格納

じバンクに格納されているため，バンク競合が発生し，データ供給を滞らせてしまう．

この問題を解決するために，各メモリバンクに対してスキュードアレイでタイルデータを格納する方式を採用する．図 4.13 にスキュードアレイ形式でのタイルデータのバンク割り当てを示す．メモリ中のタイルデータの位置によって，各バンクへの格納位置をずらし，バンク競合が発生しないようにする．このように，データを複数バンクにずらして格納することで，ラスタ走査順の 1 次元画素配列で構成されるラインデータと，2 次元スモールタイルで構成されるタイルデータの両方のデータ転送を滞りなく行うことが可能となる．

4.2.2 アドレス変換機構

通常の命令はデータがラスタ形式で格納されていることを前提としており、タイル形式での格納を想定していない。そのため、タイル形式で格納されたデータに対し、必要なデータ列の先頭アドレスを送っても、目的と違うデータを参照してしまう。そのため、ラスタ形式のアドレスからタイル形式での格納に対応したアドレスに変換する必要がある。ラスタアドレスからタイルアドレスへの変換はアドレスビット列の並び替えで可能である。参照データの横幅が 2048 バイトの場合の例を図 4.14 に示す。ラスタアドレスの 0 から 2 ビット目がラスタ形式で格納されているデータの 8 バイト分を、11 から 12 ビット目が縦方向の 4 行分を管理しているため、アドレスの 11 から 12 ビット目を 3 から 4 ビット目に挿入することで、タイル形式に対応したアドレスに変換することができる。このように、参照データの横幅が 2 のべき乗ならば、複雑な処理をすることなく、ラスタアドレスからタイルアドレスへの変換を行うことができる。

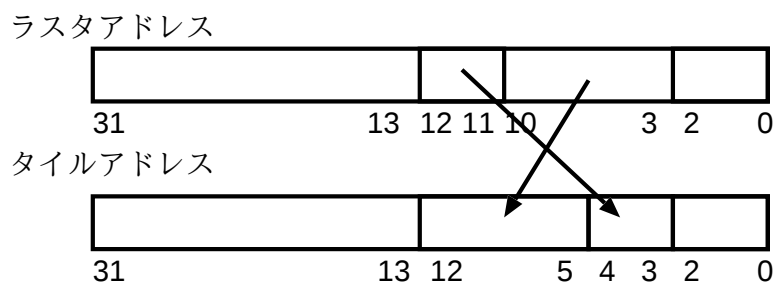


図 4.14: ビット入れ替えによるアドレス変換

4.2.3 タイル/ラインデータの転送処理

図 4.15 にアドレス変換部の詳細を示す。先頭アドレスに対し、タイルサイズに対応した 2 種類のオフセット (8x4 データに対しては 8 と 15) を足し合わせ、計 3 種類のアドレスに対してビット入れ替え処理を行う。その後、アドレス制御部にて、先頭アドレスを用いて 4 つのバンクにそれぞれ対応するアドレスを転送する。

図 4.16 にキャッシュヒット/ミスの判定機構を示す。アドレス変換部によって変換されたアドレスを各バンクに転送し、タグの一致比較を行う。キャッシュヒット時には、データバンク内でアドレスインデックスによ

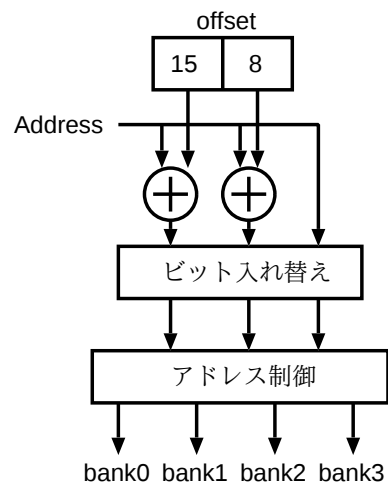


図 4.15: アドレス変換部詳細

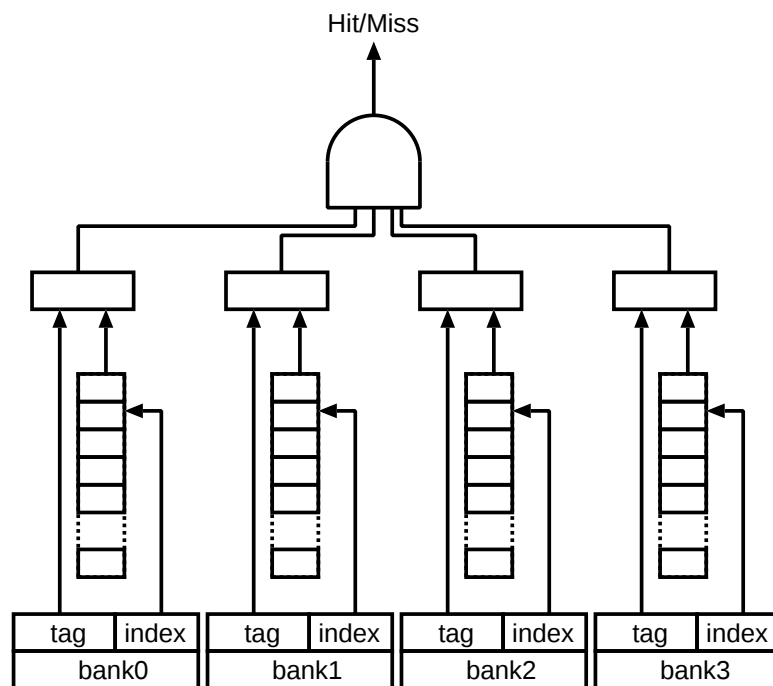


図 4.16: キャッシュヒット/ミス判定

て指定されたデータをアライメント解除部に転送し，必要データの抽出を行う．キャッシュミス時には，各バンク内の不足しているデータアドレスを順番に下位メモリに出力し，キャッシュヒットになるまで下位メモリからデータの読み込みを行う．

図 4.17 にラインデータ読み出し時の例を，図 4.18 にタイルデータ読み出しの例を示す．ラインデータ読み出しの際には，先頭アドレスで指定された同一のアドレスを各バンクに出力し，そのアドレスインデックスを用いてヒットミスの判定とデータ出力を行う．タイルデータ読み出しの際には，アドレスインデックスを 1 インクリメントした値を隣接するバンクに出力し，ヒットミス判定とデータ出力を行う．

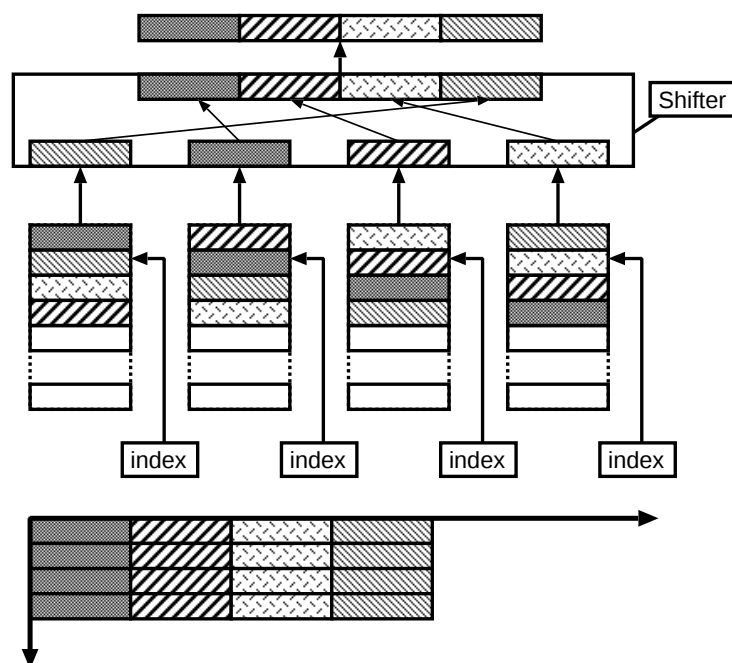


図 4.17: ラインデータ読み出し

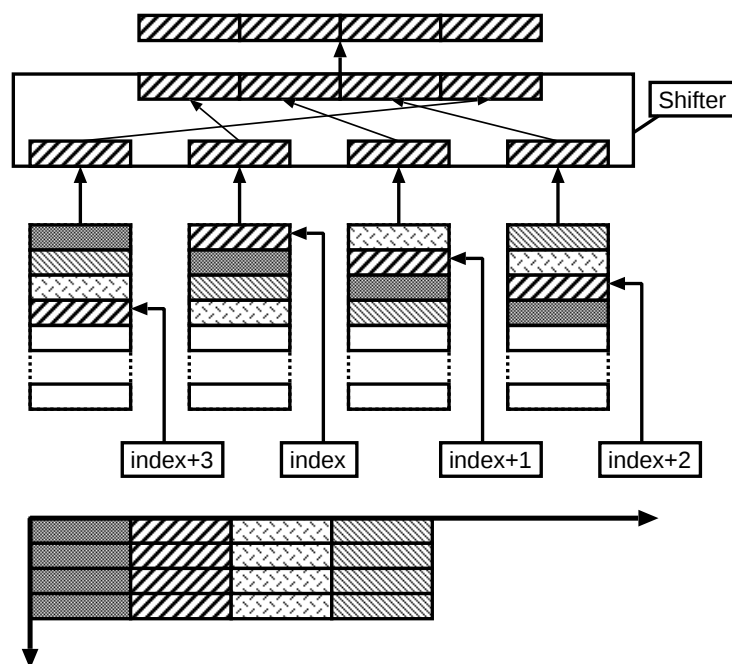


図 4.18: タイルデータ読み出し

4.3 2次元データ転送と転置処理の高速化

提案キャッシュではライン単位での転送だけでなく、タイル単位での転送にも対応している。従来キャッシュではメモリへのアクセス方法はライン単位の1次元データに限られていたため、行列転置処理を複数のunpack命令で実現せざるを得なかった。しかし、提案キャッシュメモリではタイル単位でのアクセスに対応しており、シャッフル命令を用いた転置が可能となる(図4.19)。具体的には、 4×4 の2次元タイルデータをひとつの256bitレジスタ(ymmレジスタ)に格納する。ymmレジスタに格納されたデータに対し、2種類の命令を用いてデータの並び替えを行う。この手法では、レジスタの入れ替えを行う必要が無く、わずか2命令2サイクルで実行可能となる。従来手法では 4×4 ブロックデータの転置に13サイクル必要だったため、6.5倍の高速化が見込める。また、AVX2で追加されたギャザー命令を使用すれば、わずか1命令で転置可能であり、ギャザー命令使用時の問題点であるデータアライメント制約による性能低下も無視できるため、より高速に処理できる可能性もある。

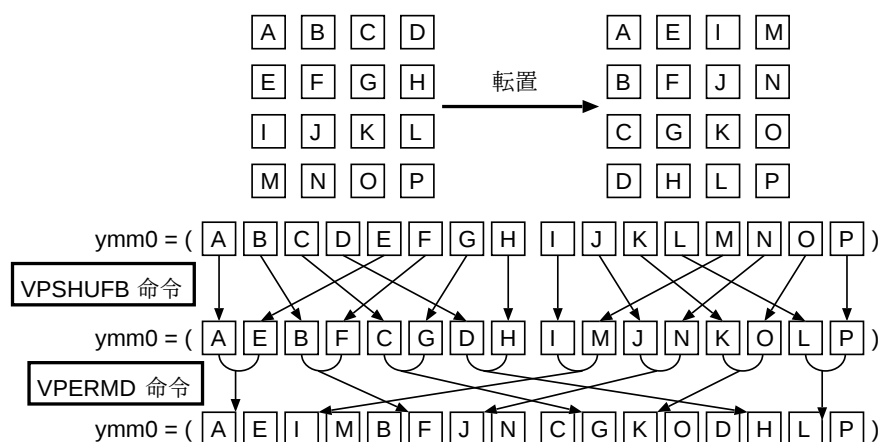


図 4.19: シャッフル命令を用いた転置処理の高速化

表 4.3 に従来キャッシュと提案キャッシュでの 4x4-32x32 のブロックデータの転置に必要なサイクル数を示す。

4x4 の転置処理時には 6.5 倍の高速化が見込めるが、8x8 以上のブロックデータに対しては高速化率が 4.5 倍となり、4x4 ブロックほどの高速化は見込めなくなった。これは、4x4 ブロックの転置処理では AVX で使用される ymm レジスタをフルで使うことが無く、それ以上のサイズのブロックに対しては ymm レジスタが効率よく使用されているからだと考えられる。行列転置処理の高速化により、2 次元 DCT 処理全体では約 1.4 倍の高速化が見込める。2 次元 DCT 処理のように一つのデータが 16bit に収まるような場合には図 4.19 に示す方法で転置処理が可能となるが、データサイズがそれ以上大きくなってしまうと、4x4 のブロックデータを ymm レジスタに格納できず、2 命令で実行することは出来ない。一つのデータが 32bit の場合には、4x4 ブロックの転置に計 4 命令必要となる。

表 4.3: 転置処理の所要サイクル数

	従来キャッシュ	提案キャッシュ
4x4	13cycle	2cycle
8x8	36cycle	8cycle
16x16	144cycle	32cycle
32x32	576cycle	128cycle

それに対して従来の転置手法では，データサイズが変わっても処理に必要な命令数は変化しないため，高速化率は3.25 倍となる．

5 評価

5.1 評価環境構築

H.264 の動画像符号化ソフトウェアである JM 参照ソフトウェアの整数画素精度動き検出部 (EPZS) にキャッシュメモリの機能シミュレータを組み込み、従来キャッシュと先行研究の Two-bank interleaved cache , Block offset mapping cache , 提案キャッシュの 4 種類で比較評価を行った . 各メモリへの読み込みサイクル数は表 5.4 のように仮定し、符号化条件は表 5.5 のように設定した .

SSE4 に搭載されている MPSADBW 命令では、水平方向の複数の SAD を並列に演算することが可能であるため、探索対象マクロブロックの水平 1 ラインがキャッシュ内に存在すればヒット、そうでなければミスとしてヒット率の評価を行った . 今回実装したシミュレータでは、連想度が 14 の間で変更可能であり、連想度が 2 以上であるセットアソシアティブキャッシュのライン入れ替えアルゴリズムには、LRU(Least Recently Used) を使用した .

表 5.4: メモリ構成

メモリ階層	容量	連想度	レイテンシ
L1 キャッシュ	1KB-32KB	1	1cycle
L2 キャッシュ	256KB	1	10cycle
主記憶	制限なし	なし	200cycle

表 5.5: 符号化条件

画像シーケンス (フレーム番号)	Horse Race(390-420) Bronze_with_credits(260-290)
入力画像サイズ	1920x1056, 30 _{fps}
フレーム数	30
エンコーダー	JM 参照ソフトウェア
シーケンス構成	M=2(IBBP...)

5.2 性能評価

図 5.20 に HorseRace 画像シーケンス，図 5.21 に Bronze_with_credits 画像シーケンスでの評価を，表 5.6 にキャッシュヒット率を示す．単純なキャッシュ性能の比較を行うために，キャッシュメモリの格納構造はダイレクトマップとした．どちらの画像シーケンスにおいても，L1 キャッシュ容量 1-32KB の全ての場合において，提案キャッシュは他のキャッシュメモリよりも高性能となった．特に低容量キャッシュにおいて大幅な性能向上が見られ，提案キャッシュを用いることで，データ転送に要する時間を従来キャッシュの約 30% 低減することが出来た．しかし，先行研究の Two-bank interleaved cache との比較では，キャッシュ容量 32KB の場合において，HorseRace 画像で約 5%，Bronze_with_credits 画像では約 15% と画像シーケンスによってばらつきがあるが，低減率はあまり高くない．Block-offset mapping cache では，1KB の時のみ通常キャッシュよりも高性能だったが，2KB 以上のキャッシュメモリにおいては従来キャッシュよりも低性能となった．このような結果になった原因として，JM 参照ソフトウェアの動き検出部がラスト走査順の処理法に基づいており，Block-offset mapping では，矩形タイルデータを一つのキャッシュメモリに格納するため，ラスト走査順の処理法では必要データがキャッシュライン境界をまたぐ確率が高いことから，データアライメント制約によりオーバーヘッドが発生したものと考えられる．

表 5.6: キャッシュのヒット率

HorseRace シーケンス						
	1KB	2KB	4KB	8KB	16KB	32KB
従来キャッシュ	43.2%	65.0%	76.2%	85.5%	94.8%	97.3%
提案キャッシュ	72.9%	82.0%	87.9%	92.7%	97.1%	98.6%
Bronze_with_credits シーケンス						
	1KB	2KB	4KB	8KB	16KB	32KB
従来キャッシュ	45.9%	60.9%	69.6%	76.7%	84.5%	91.4%
提案キャッシュ	69.2%	76.4%	81.8%	86.5%	91.3%	94.5%

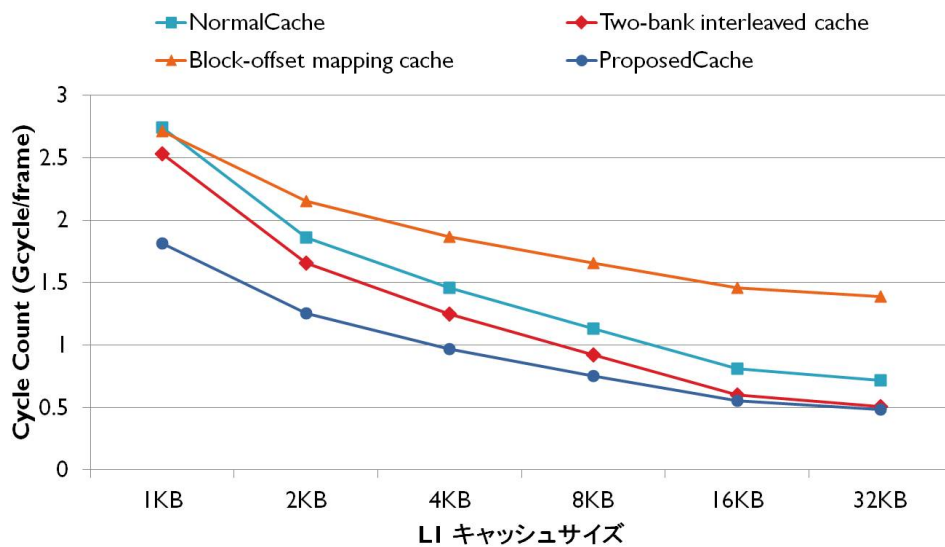


図 5.20: HorseRace 画像での評価

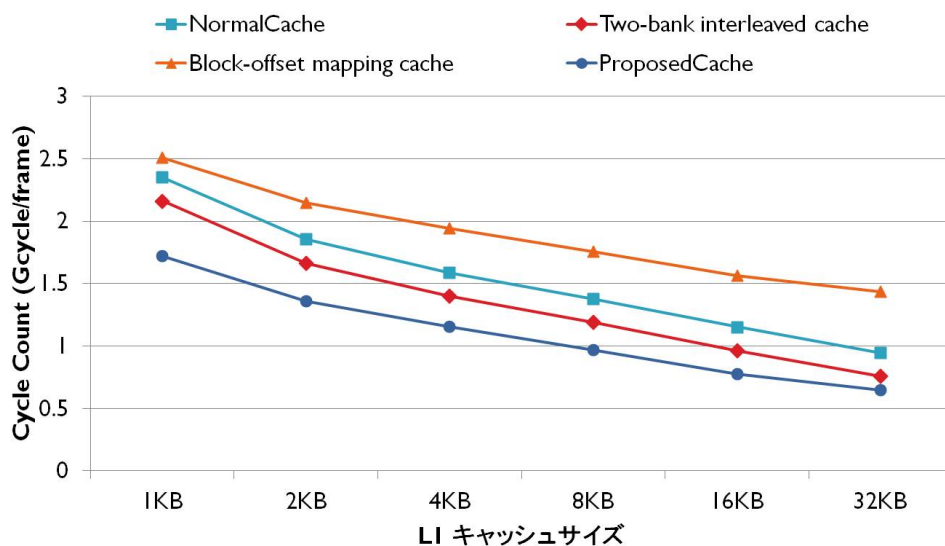


図 5.21: Bronze with credits 画像での評価

予備評価として、キャッシュラインサイズが4バイトから128バイトの間で、動き検出処理中でキャッシュライン境界を交差する確率と、参照画素1ラインのアクセスにかかる平均サイクル数を求めた(表5.7)。Block-offset mapping cache や提案キャッシュにおいては、データが8x4の2次元データで格納されるため、キャッシュライン境界をまたぐ確率は約60%であり、データアライメント制約によるオーバーヘッドが非常に大きい。従来キャッシュやTwo-bank interleaved cache においても、キャッシュライン境界をまたぐ確率は16%、平均アクセスサイクルは1.6回となり、データアライメント制約によるオーバーヘッドが多い事が分かる。この結果から、タイルデータ割り当てから必要な位置のラインデータを、データアライメント制約解除してアクセス可能とする仕組みの性能向上への寄与が非常に高いことが分かる。また、提案キャッシュでは容量が小さい場合においてもヒット率が高いことから、タイルデータ割り当てを用いることで、キャッシュミスによる性能低下を大幅に減らせる事が分かる。

表 5.7: キャッシュライン境界交差率と平均アクセスサイクル

HorseRace シーケンス						
キャッシュラインサイズ	4byte	8byte	16byte	32byte	64byte	128byte
境界をまたぐ確率	85.3%	57.6%	32.4%	16.1%	7.9%	3.9%
平均アクセスサイクル	7.3	3.8	2.3	1.6	1.3	1.2
Bronze_with_credits シーケンス						
キャッシュラインサイズ	4byte	8byte	16byte	32byte	64byte	128byte
境界をまたぐ確率	87.7%	62.3%	35.8%	17.8%	8.7%	4.2%
平均アクセスサイクル	7.4	4.0	2.4	1.7	1.3	1.2

図5.22 に提案キャッシュで連想度を変更した場合の評価を示す。今回実装した評価環境では連想度が1～4まで変更可能であるため、ダイレクトマップ、2-way セット、4-way セットの計3種類で比較評価を行った。評価の結果、連想度を上げるとに性能の向上が見られたが、キャッシュ容量が大きくなるにつれて性能差があまり見られなくなった。キャッシュメモリの連想度を増やすことによって、競合性のキャッシュミスはある程度回避することができる。動画像符号化においても、複数フレームを参照するBピクチャの符号化時には、特に競合性ミスが発生しやすくなるため、連想度を上げることによりキャッシュ性能を向上させることが可能と

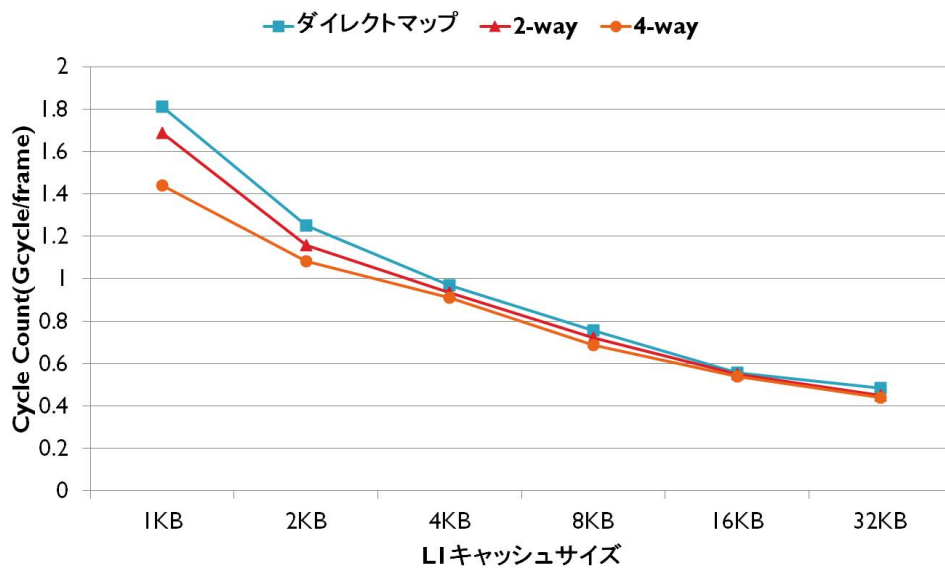


図 5.22: 連想度別評価

なる。

5.3 ハードウェア規模評価

今回、前述の提案キャッシュメモリの設計を行い、ハードウェア規模の検証を行った。データ容量が4バンク合計1KBで、機能や構成については4章で述べた通りのものを設計した。さらに、提案キャッシュに対し、次節で述べるタグメモリの容量削減手法を適用したキャッシュメモリに対しても同様に設計を行い、ハードウェア規模の検証を行った。

5.3.1 タグメモリの容量削減

本研究では、4章で述べた提案キャッシュに対して、後述するタグメモリの容量削減手法を用いることで、ハードウェア規模の低減を図る。4章で述べた設計では、提案キャッシュのタグメモリは、各データバンクの1ライン(8バイト)毎にタグを持つ構成を採っている(図5.23(a))。この構成では、必要データの有無を判定するタグの一致比較回路が容易に設計できる代わりに、タグメモリの容量が従来の約4倍になってしまうという欠

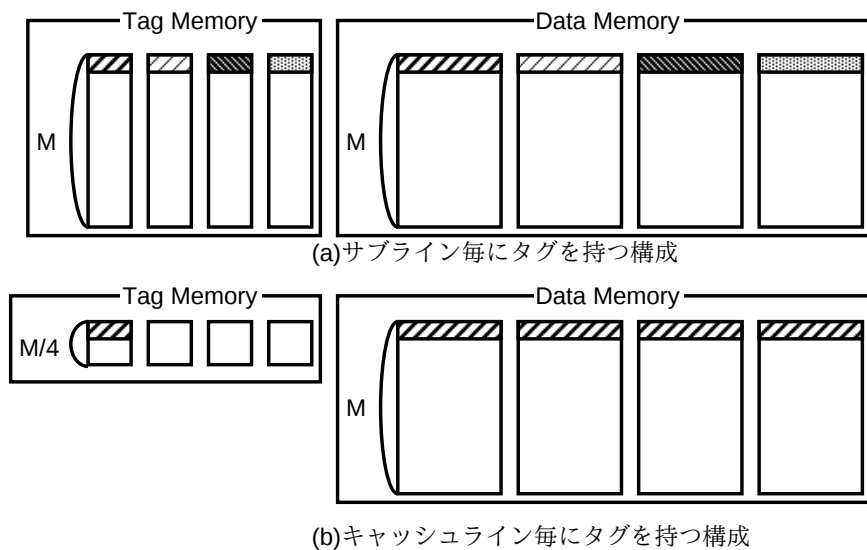


図 5.23: タグメモリの容量削減

点がある．従来と同様にキャッシュライン (32 バイト) 毎にタグを持つ構成に変更することで，タグメモリの容量を削減することが可能である (図 5.23(b))．しかしこの構成では，タグの一致比較回路が複雑となり，ハードウェア規模やレイテンシの増大を引き起こしてしまう可能性がある．

5.3.2 結果

ハードウェア規模は 2 入力 NAND に換算したゲート数で算出した．通常キャッシュ，提案キャッシュ，提案キャッシュに対して前節のタグメモリ容量削減手法を適用したものの計 3 種類の間で比較評価を行った．評価結果を表 5.8 に示す．評価の結果，提案キャッシュは通常キャッシュの約 1.25 倍のハードウェア規模となった．メモリ部分を除く周辺回路は通常キャッシュと比べ，大きく増加してしまった．しかし周辺回路と比べ，メモリ部分が非常に大きいため，全体のハードウェア規模ではそれほど影響はないと考えられる．さらに，提案キャッシュに対してタグ容量削減手法を適用することで，メモリ部分の規模を大幅に削減することができ，キャッシュメモリ全体のハードウェア規模を約 1.1 倍にまで抑えることが可能となる．代わりに周辺回路のハードウェア規模が増大しているが，ハードウェア全体の規模は大きく削減可能であるため，ハードウェ

表 5.8: ハードウェア規模評価

	周辺回路	メモリ部分	全体	動作周波数
通常キャッシュ	2334	71121	73455	500MHz
提案キャッシュ	6521	83897	90418	100MHz
提案キャッシュ (タグ容量削減後)	8244	74713	82417	100MHz

ア規模の面では問題が無い。しかし，提案キャッシュはシフタやアドレス変換器などの追加により，従来キャッシュと比べて低速なため，設計の最適化が必要となる。

6 あとがき

本論文では，SIMD 演算処理におけるデータ供給ボトルネックを改善する新構成のキャッシュメモリを提案した．

提案したキャッシュメモリは，SIMD 演算処理時のボトルネックを解消するデータアライメント制約解除機構と，マルチメディア処理や行列計算処理などに対し，タイルデータ割り当てにより 2 次元ブロックデータのアクセスにおける空間的局所性を発揮できる構成となっている．また，ラスタ走査順の 1 次元画素配列で構成されるラインデータと，2 次元ブロックデータで構成されるタイルデータの両方のデータアクセスに対応可能としている．動画像符号化ソフトウェア内に提案キャッシュメモリと従来キャッシュメモリの機能シミュレータを組み込み，比較評価を行ったところ，データ転送に要するサイクル数を約 30%削減できた．また，行列転置処理においては，タイルデータの転送とシャッフル命令を組み合わせることにより，4x4 ブロックデータの転置処理が 6.5 倍高速化できた．しかし，データアライメント制約解除機構などの追加ハードウェアにより，従来キャッシュと比べてハードウェア規模やレイテンシが増大してしまうという問題がある．特に，読み込みレイテンシの増加は CPU サイクルタイムの増加を引き起こすため，必要データを含むデータラインから必要データのみを抜き出す部分の処理を，SIMD データパスに移すなどにより，レイテンシの増大を防ぐ必要がある．また，提案キャッシュの実効性能を評価するために，専用命令セットの制定とそれに対応するコンパイラを作成する必要がある．

謝辞

本研究を遂行するにあたり，日頃から御指導，御助言を頂きました近藤利夫教授，佐々木敬泰助教に感謝いたします．また，研究に協力していただいた計算機アーキテクチャ研究室の方々に感謝の意を表します．

参考文献

- [1] Chang, Hoseok, and Wonyong Sung. "Efficient vectorization of SIMD programs with non-aligned and irregular data access hardware." Proceedings of the 2008 international conference on Compilers, architectures and synthesis for embedded systems. ACM, 2008.
- [2] Huang, Yu-Wen, et al. "Analysis, fast algorithm, and VLSI architecture design for H. 264/AVC intra frame coder." Circuits and Systems for Video Technology, IEEE Transactions on 15.3 (2005): 378-401.
- [3] Lo, Wing-Yee, et al. "Improved SIMD architecture for high performance video processors." Circuits and Systems for Video Technology, IEEE Transactions on 21.12 (2011): 1769-1783.
- [4] X264 FreeH.264 /AVC Encoder [Online]. Available: <http://www.videolan.org/developers/x264.html>
- [5] 高橋大介. "共有メモリ型並列計算機における並列 FFT のブロックアルゴリズム." 情報処理学会論文誌 43.4 (2002): 995-1004.
- [6] Alvarez, Mauricio, et al. "Performance impact of unaligned memory operations in SIMD extensions for video codec applications." Performance Analysis of Systems & Software, 2007. ISPASS 2007. IEEE International Symposium on. IEEE, 2007.
- [7] Lam, Monica D., Edward E. Rothberg, and Michael E. Wolf. "The cache performance and optimizations of blocked algorithms." ACM SIGARCH Computer Architecture News. Vol. 19. No. 2. ACM, 1991.
- [8] Soo-Ik, C. H. A. E. "Cache optimization for H. 264/AVC motion compensation." IEICE transactions on information and systems 91.12 (2008): 2902-2905.

- [9] Kim, Ju-Hyun, Gyoung-Hwan Hyun, and Hyuk-Jae Lee. "Cache organizations for H. 264/AVC motion compensation." Embedded and Real-Time Computing Systems and Applications, 2007. RTCSA 2007. 13th IEEE International Conference on. IEEE, 2007.

A 実験手順

A.1 シミュレーションの実装手順

- epzs.c にある関数 `computeSad` と `computeBiPredSad` 内から別途作成した評価用関数を呼び出す．`computeSad` 関数は P ピクチャでの SAD 計算，`computeBiPredSad` 関数は B ピクチャでの SAD 計算を行う関数である．
- 評価には，`blocksize_x`(マクロブロックの横幅)，`blocksize_y`(マクロブロックの縦幅)，`cand_x`(参照ブロックの左上の x 座標)，`cand_y`(参照ブロックの左上の y 座標) の計 4 種類の変数を使用する．
- 評価関数では，キャッシュメモリのタグビット，有効ビット，LRU ビットを記憶する配列を用意し，参照した画素アドレスの各要素を各配列に記憶する．
- 画素アドレスは参照ブロックの x, y 座標 (`cand_x`, `cand_y`) と対象画像のサイズから導出する．
- 画素アドレスとキャッシュ配列間で 1 画素ずつ一致比較を行い，マクロブロックの水平 1 ライン全てがキャッシュ配列内に存在すればヒット，1 画素でも欠けている部分があればミスとする．
- L1 キャッシュ，L2 キャッシュ，主記憶の 3 つの階層に対してアクセス回数を記憶し，データ転送に要するサイクル数を導出する．