動画像デコーディングのIntelおよびARMマルチコア上での並列処理の評価

和気 珠実[1,a)]　飯塚 修平[1]　見神 広紀[1]　木村 啓二[1]　笠原 博徳[1]

概要：本稿では, マルチコアプロセッサを用いて動画像デコーディング処理の高速化を実現する手法として2種類の並列化手法について性能評価を行った. 1つ目の並列化手法は並列化対象ループにループスキューイング/ループインターチェンジを適用する手法, 2つ目の並列化手法はwave-front手法を適用する手法であり, どちらの場合もマクロブロック間の依存関係を満たしつつこれらの間の並列性を利用することで並列処理が可能となる. 評価に用いる動画像コーデックは, MPEG2と比較して約2倍の符号化効率を持ちワンセグ放送等に用いられているH.264/AVCと, H.264/AVCと同等の品質を持ちYoutube等でも採用されている動画規格であるWebMのビデオコーデックVP8である. これらの規格により動画像デコーディングを行うプログラムに対して, 上記2つの並列化手法をそれぞれ適用した. Snapdragon APQ8064 Krait 4コアを搭載したNexus7上で評価を行った結果, ループスキューイング/ループインターチェンジ手法で並列化した場合, 並列化箇所のみで逐次実行に比べ3コアで1.33倍速度向上し, その一方でwave-front手法では3コアで2.86倍の速度向上が得られた. 同様にIntel(R) Xeon(R) CPU X5670プロセッサを搭載したマシンで評価を行った結果, ループスキューイング/ループインターチェンジ手法で並列化した場合, 並列化箇所のみで逐次実行に比べ6コアで1.82倍速度向上し, 一方でwave-front手法では6コアで4.61倍の速度向上が得られた.

1. はじめに

近年, スマートフォンやタブレット端末等の急速な普及に伴い動画閲覧の需要が増大してきている. 動画閲覧には高い品質やリアルタイム性が求められている一方で, 品質を高めることで処理負荷の増大が問題となっている. そのため, 高負荷の動画像処理を低消費電力で実現可能なプロセッサが求められている. 従来は性能向上のために動作周波数を上げることで対応していたが，周波数を上げることにより消費電力や発熱量の増大といった問題が挙げられるため, 現在はマルチコアプロセッサによる処理の高速化への期待が高まっている. マルチコアプロセッサを用いることで，プロセッサ全体の処理性能を高めながら消費電力や発熱量を低く抑えることが見込まれている.

高速化のためには専用デバイスなどのハードウェアによる手法も考えられるが, ハードウェアの開発には高度な技術が必要な上に, サポートが求められる動画像コーデックが多様化していることを考えると, マルチコアプロセッサを用いたソフトウェアによる処理の高速化の方が有望である.

ソフトウェアによる並列化には様々な手法が提案されており, 並列化の対象とされているプログラムの構造によって最適な手法は変わってくる. 動画像デコーディングにおいてはその依存関係から, コア間で同期処理を必要とするwave-front手法 [1] や, wave-front手法よりもコンパイラによる自動生成をしやすい手法としてループスキューイング/ループインターチェンジ [2][3] を適用する手法などが提案されている. そこで本稿では，動画形式WebMのビデオコーデックVP8[4] とH.264/AVC[5] を対象として，一般的に広く用いられている並列化手法であるループスキューイング/ループインターチェンジ手法とwave-front手法の2種類の並列化手法をOpenMP[6] を用いて実装し, その性能比較をIntel[7] およびARM[8] マルチコアプラットフォーム上で行う.

本稿では, 第2章でWebMデコーダとH.264/AVCデコーダの概要について，第3章で2種類の並列化手法について, 第4章で性能評価結果について述べ，最後に第5章で本稿のまとめを述べる.

2. WebMおよびH.264/AVCデコーダ

本章では，動画像デコーダーであるWebMおよび

[1] 早稲田大学
Waseda University.
[a)] waketama@kasahara.cs.waseda.ac.jp

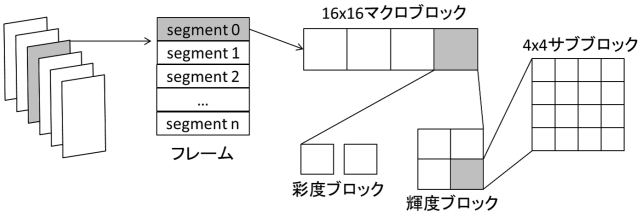

図 1 WebM デコーダのデータ構造

H.264/AVC について述べる.

2.1 WebM および H.264/AVC の概要

WebM は Google 社が開発した動画形式であり, 従来より DVD 等で広く用いられてきた MPEG2[9] と比較して高圧縮かつ高品質であることから現在 Youtube 等でも採用されている規格である. WebM デコーダのデータ構造は図 1 に示すように, フレーム層, セグメント層, マクロブロック層, ブロック層, というように階層的なデータ構造になっている [4][5][10]. 本稿で並列化 [11] を行ったのはマクロブロック層についてである. 並列化手法については 3.2 節と 3.3 節で詳しく述べる.

H.264/AVC は, ITU-T と ISO/IEC の 2 つの標準化機関が合同で開発を進めた動画圧縮規格であり, WebM 同様, MPEG2 と比較して高い圧縮効率を誇るが, 従来の MPEG2 では計算複雑度から採用されなかった記述も多く取り入れられているため計算量は膨大となっている [12]. H.264/AVC のデータ構造は WebM のものとほぼ同様である. なお, 本稿では H.264/AVC デコーダの並列化を可能とするために, 先行してビット列を走査することで各行の区切れ目を把握するプレスキャン処理を追加している.

WebM や H.264/AVC のような動画像コーデックは高い圧縮率が得られる反面, 計算量が膨大であるためデコード時間が長くなってしまう. 計算時間が膨大になってしまう主な原因は, 圧縮のため同フレーム内や他フレームからの情報を参照して予測を行う予測処理や, ブロック間の歪みを除去するフィルター処理である. これらの処理の計算時間を並列化により短縮することでスムーズな動画閲覧の実現が期待できるため, 本稿では予測処理について並列化を行った.

2.2 WebM および H.264/AVC のデコード処理フロー

WebM および H.264/AVC のデコード処理フローは図 2 のようになっており, デコード処理は大きく分けてエントロピー復号, 逆量子化・逆 DCT, 画面内予測, 動き補償予測, デブロッキングフィルタに分けられる [5]. まずエントロピー復号でエンコーダにより符号化された情報を復号化し, 逆量子化・逆 DCT で復号化された量子化 DCT 係数を画素空間データへ復元する. 画面内予測では同フレーム内のマクロブロックを利用した予測処理を実施し, 動き補償

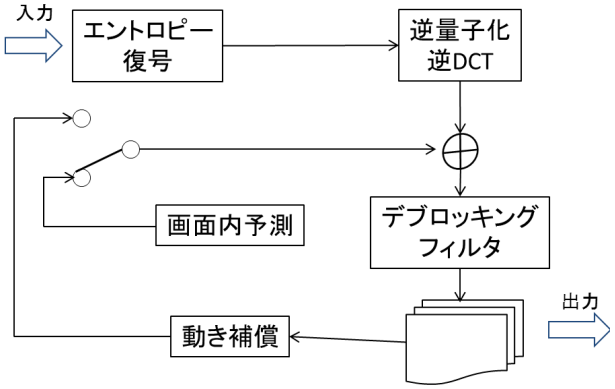

図 2 WebM および H.264/AVC のデコード処理フロー

予測では他のフレームの周囲ブロックから動きベクトルを予測し, 参照フレームと動きベクトルを用いて予測処理を行う. デブロッキングフィルタでは, デコード処理の際に生じたブロックノイズの影響を除くためにブロック間の平滑化を行う.

3. WebM および H.264/AVC の並列化

本章では, WebM および H.264/AVC のマクロブロック階層の依存関係と, マクロブロック間の依存関係を考慮した 2 種類のマクロブロックレベルでの並列化手法について述べる.

3.1 WebM および H.264/AVC のマクロブロック階層の依存関係

WebM および H.264/AVC は, 2.1 節で述べたような階層的なデータ構造を持つ. このうちデコード処理の大部分を占める予測処理やフィルター処理が割り当てられる単位はマクロブロックレベルであるため, 本稿ではマクロブロックレベルでの並列化を行った.

両規格ともマクロブロック間には図 3 のような依存関係があるため, 並列化する際にはこれらを十分考慮した上で並列化をしなければならない. 具体的には, あるマクロブロックの予測処理とフィルター処理を行うためには, そのマクロブロックの左, 左上, 上, 右上のマクロブロックに依存関係があるため, この 4 つのマクロブロックの処理が終わっている必要がある. このような依存関係を満たしながら並列処理を行うために一般的に知られている 2 つの手法で並列化を行った. 1 つ目は並列化対象のマクロブロックループにループスキューイング/ループインターチェンジを施した手法で, 3.2 節にて詳しく述べる. 2 つ目は並列化対象のマクロブロックループを wave-front 手法で並列化する方法で, 3.3 節で詳細について説明する.

3.2 ループスキューイング/ループインターチェンジ手法による並列化

WebM および H.264/AVC のマクロブロック間には 3.1

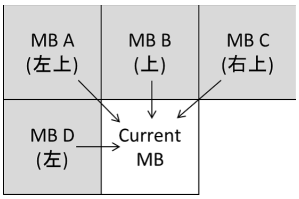

図 3 WebM および H.264/AVC のマクロブロック間の依存関係

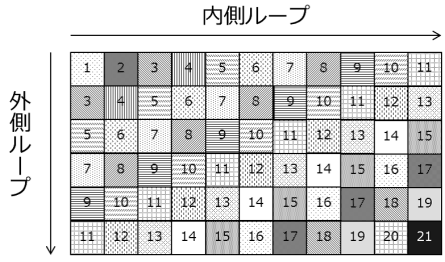

図 4 並列化適用前のマクロブロックの構成イメージ

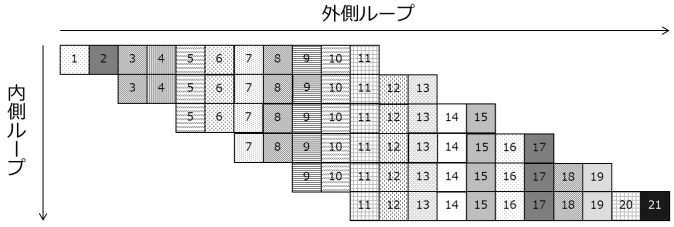

図 5 ループスキューイング/ループインターチェンジ手法適用後のマクロブロックの構成イメージ

節で述べたような依存関係がある．これらの依存関係を考慮すると, 図 4 における同番号のマクロブロックは並列に実行することが可能であり, 並列化ループの単純な二分割や行ごとの交互実行では依存関係を満たすことができないが, 処理実行順序に工夫を施すことにより並列実行が可能となる．ここで同番号のマクロブロックの並列実行を可能とするために並列化対象ループにループスキューイング/ループインターチェンジを適用した図が図 5 である．これにより並列処理可能箇所が縦方向に並ぶため, 同番号の振られているマクロブロックをコアごとに割り当てて並列実行することが可能となる．なお, 本稿では WebM デコーダのデータ構造に則し, 同時に 8 行以上は実行しない状況下で評価を行った．

3.3 wave-front 手法による並列化

図 4 において各コアが各マクロブロック行の処理を担当する手法として wave-front 手法を適用した．マクロブロック行ごとに処理の進行度を保持させ常に行間でフラグ変数の送受信等の手段により互いの進行度を伝え合い, 3.1 節で説明した依存のある 4 つのマクロブロックの処理が終わっていたら担当のマクロブロックの処理を行う．具体的には, WebM や H.264/AVC の場合は 1 つ下の行のデコード処理を 2 イタレーション遅らせることによりマクロブロック間の依存関係を守りつつ並列実行が可能となる．

表 1 Nexus7 性能

CPU	Snapdragon APQ8064 Krait
CPU core	4 cores
CPU Frequency	1.5GHz
L1 cache	32KB
L2 cache	2MB
OS	AndroidOS 4.3

表 2 HA8000/TS20 性能

CPU	Intel Xeon CPU X5670
CPU core	12 cores
CPU Frequency	2.93GHz
L1 D-cache	32KB
L1 I-cache	32KB
L2 cache	256 KB
L3 cache	12MB for 6 cores

4. 性能評価

本章では, 3.2 節および 3.3 節で述べたマクロブロックレベルの並列化手法を用いた WebM および H.264/AVC デコーダの性能評価結果について述べる．

4.1 評価環境

本節では，WebM および H.264/AVC の性能評価を行う際に用いた評価環境について述べる．

4.1.1 Nexus7

本稿で評価に用いた携帯端末は，Qualcomm 社による ARM 命令セットのコアを 4 コア持つマルチコアプロセッサを搭載した Nexus7 2013 年モデル (以下 Nexus7) である．Nexus7 の諸元を表 1 に示す [13].

4.1.2 HA8000/TS20

本稿で評価に用いたサーバーマシンは, 6 コアの Intel(R) Xeon(R) CPU X5670 プロセッサを 2 基搭載した HA8000/TS20 である．HA8000/TS20 の諸元を表 2 に示す [14].

4.2 評価方法

WebM および H.264/AVC デコーダの評価にはファイル出力をしないオプションを用いた．また WebM の実行時には, VP8 の仕様に合わせエンコードの時点で partition の数を指定して作成した WebM ファイルを入力とする．入力データはニュース番組の影像の一部で, シーンの移り変わりなど動きの多い画像である．入力データの詳細を表 3 に示す．

4.3 Nexus7 における性能評価結果

本節では, WebM デコーダおよび H.264/AVC デコーダを Nexus7 上で性能評価した結果を示す．

表 3 WebM および H.264/AVC 評価に用いた入力ファイル

ファイル形式	webm および 264
フレーム数	1049
解像度	1920x1080 および 1280x720

表 4 ループスキューイング/ループインターチェンジ手法による Nexus7 における WebM デコーダの実行時間 [秒]

	1080p		720p	
	予測処理	デコード全体	予測処理	デコード全体
original	33.88	103.93	18.10	51.67
1 コア	57.28	132.77	27.95	63.16
2 コア	33.58	108.92	16.41	51.58
3 コア	26.96	102.34	13.58	48.82

表 5 wave-front 手法による Nexus7 における WebM デコーダの実行時間 [秒]

	1080p		720p	
	予測処理	デコード全体	予測処理	デコード全体
original	33.88	103.93	18.10	51.67
1 コア	35.31	105.22	18.63	51.98
2 コア	17.53	87.42	9.34	42.66
3 コア	11.86	81.73	6.36	39.69

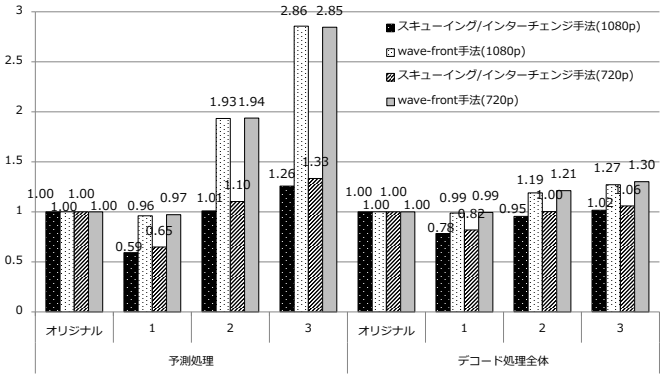

図 6 Nexus7 における WebM デコーダの速度向上率

まず, WebM デコーダをループスキューイング/ループインターチェンジ手法を用いて並列化した場合のコア数に対する処理時間を表 4, wave-front 手法を用いた場合の処理時間を表 5 に示す. また, その際の両者の速度向上率を図 6 に示す. 本稿で並列化した予測処理において, オリジナル版と比較しループスキューイング/ループインターチェンジ手法では 3 コアで 1.33 倍, wave-front 手法では 3 コアで 2.86 倍の速度向上となった.

次に, H.264/AVC デコーダをループスキューイング/ループインターチェンジ手法を用いて並列化した場合のコア数に対する処理時間を表 6, wave-front 手法を用いた場合の処理時間を表 7 に示す. また, その際の両者の速度向上率を図 7 に示す. 本稿で並列化した予測処理において, オリジナル版と比較しループスキューイング/ループインターチェンジ手法では 3 コアで 1.47 倍, wave-front 手法では 3 コアで 1.94 倍の速度向上となった.

表 6 ループスキューイング/ループインターチェンジ手法による Nexus7 における H.264/AVC デコーダの実行時間 [秒]

	1080p			720p		
	(prescan)	予測	全体	(prescan)	予測	全体
original	なし	170.24	256.51	なし	75.30	111.76
1 コア	31.03	236.56	307.43	14.05	103.19	133.27
2 コア	31.03	145.17	216.14	14.05	64.76	94.79
3 コア	31.03	116.14	187.12	14.05	52.25	82.31

表 7 wave-front 手法による Nexus7 における H.264/AVC デコーダの実行時間 [秒]

	1080p			720p		
	(prescan)	予測	全体	(prescan)	予測	全体
original	なし	170.24	256.51	なし	75.30	111.76
1 コア	31.03	198.10	269.46	14.05	89.20	119.34
2 コア	31.03	114.37	185.26	14.05	51.12	81.19
3 コア	31.03	87.89	158.85	14.05	39.99	70.09

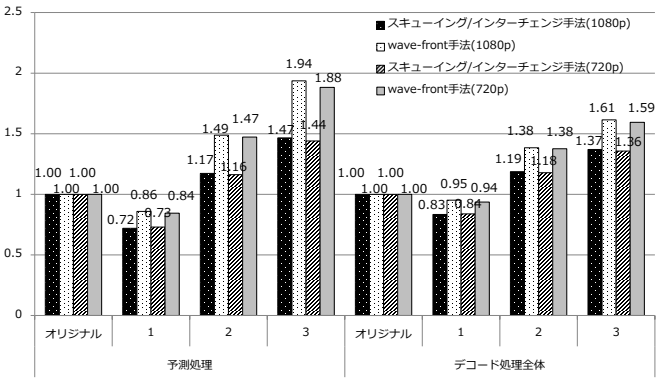

図 7 Nexus7 における H.264/AVC デコーダの速度向上率

4.4 HA8000/TS20 における性能評価結果

本節では, WebM デコーダおよび H.264/AVC デコーダを HA8000/TS20 上で性能評価した結果を示す.

まず, WebM デコーダをループスキューイング/ループインターチェンジ手法を用いて並列化した場合のコア数に対する処理時間を表 8, wave-front 手法を用いた場合の処理時間を表 9 に示す. また, その際の両者の速度向上率を図 8 に示す. 本稿で並列化した予測処理において, オリジナル版と比較しループスキューイング/ループインターチェンジ手法では 6 コアで 1.82 倍, wave-front 手法では 6 コアで 4.61 倍の速度向上となった.

次に, H.264/AVC デコーダをループスキューイング/ループインターチェンジ手法を用いて並列化した場合のコア数に対する処理時間を表 10, wave-front 手法を用いた場合の処理時間を表 11 に示す. また, その際の両者の速度向上率を図 9 に示す. 本稿で並列化した予測処理において, オリジナル版と比較しループスキューイング/ループイン

表 8 ループスキューイング/ループインターチェンジ手法による HA8000/TS20 における WebM デコーダの実行時間 [秒]

	1080p		720p	
	予測処理	デコード全体	予測処理	デコード全体
original	11.53	36.76	6.09	18.19
1 コア	12.87	38.13	6.74	18.84
2 コア	9.45	34.70	4.85	16.96
4 コア	6.99	32.22	3.51	15.62
6 コア	6.66	31.91	3.35	15.48

表 9 wave-front 手法による HA8000/TS20 における WebM デコーダの実行時間 [秒]

	1080p		720p	
	予測処理	デコード全体	予測処理	デコード全体
original	11.53	36.76	6.09	18.19
1 コア	11.95	37.19	6.20	18.29
2 コア	6.22	31.42	3.34	15.44
4 コア	3.36	28.56	1.84	13.95
6 コア	2.50	27.79	1.38	13.45

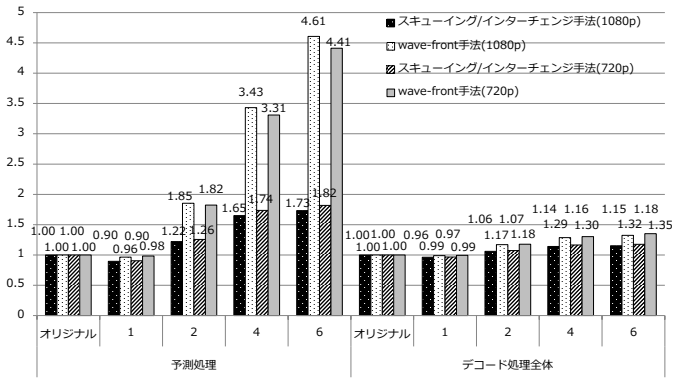

図 8 HA8000/TS20 における WebM デコーダの速度向上率

表 10 ループスキューイング/ループインターチェンジ手法による HA8000/TS20 における H.264/AVC デコーダの実行時間 [秒]

	1080p			720p		
	(prescan)	予測	全体	(prescan)	予測	全体
original	なし	30.99	46.44	なし	11.35	17.04
1 コア	4.99	42.09	55.67	2.27	16.92	22.04
2 コア	4.99	26.13	39.73	2.27	10.82	15.96
4 コア	4.99	17.33	30.93	2.27	7.32	12.46
6 コア	4.99	14.19	27.81	2.27	6.16	11.29

ターチェンジ手法では 6 コアで 2.18 倍, wave-front 手法では 6 コアで 2.60 倍の速度向上となった.

4.5 各並列処理手法の比較

Wave-front 手法による並列化では, WebM デコーダでは Nexus7 で 3 コアで 2.86 倍の速度向上, HA8000/TS20 で 6 コアで 4.61 倍の速度向上が得られた. また, H.264/AVC デコーダでは Nexus7 で 3 コアで 1.94 倍の速度向上,

表 11 wave-front 手法による HA8000/TS20 における H.264/AVC デコーダの実行時間 [秒]

	1080p			720p		
	(prescan)	予測	全体	(prescan)	予測	全体
original	なし	30.99	46.44	なし	11.35	17.04
1 コア	4.99	37.69	51.44	2.27	14.20	19.30
2 コア	4.99	22.48	36.23	2.27	8.94	14.07
4 コア	4.99	14.33	28.12	2.27	5.93	11.09
6 コア	4.99	11.91	25.65	2.27	5.11	10.26

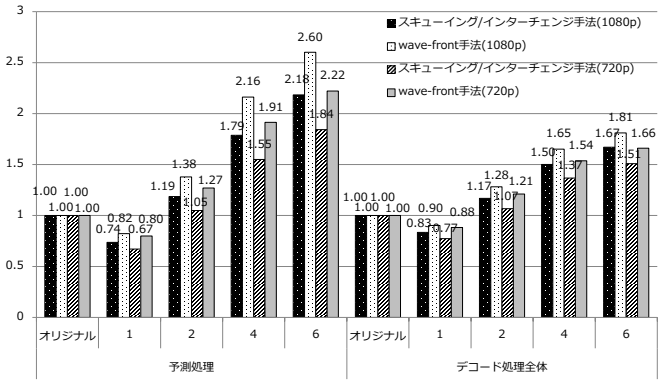

図 9 HA8000/TS20 における H.264/AVC デコーダの速度向上率

HA8000/TS20 で 6 コアで 2.60 倍の速度向上が得られた. 一方で, ループスキューイング/ループインターチェンジ手法による並列化では, WebM デコーダでは Nexus7 で 3 コアで 1.33 倍の速度向上, HA8000/TS20 で 6 コアで 1.82 倍の速度向上にとどまった. 同様に H.264/AVC デコーダにおいても, Nexus7 で 3 コアで 1.47 倍の速度向上, HA8000/TS20 で 6 コアで 2.18 倍の速度向上にとどまり, ループスキューイング/ループインターチェンジ手法による並列化では, wave-front 手法による並列化ほどの速度向上が得られなかった.

WebM では, ループスキューイング/ループインターチェンジを施すことでオリジナル版と比較し Nexus7 で最大約 24 秒, HA8000/TS20 で最大約 1.3 秒処理時間が伸びた. また, H.264/AVC ではループスキューイング/ループインターチェンジにより Nexus7 で最大約 66 秒, HA8000/TS20 で最大約 11 秒処理時間が伸びた.

H.264/AVC では並列化のためにプレスキャン処理を追加しているため, プレスキャン処理分の処理時間の伸びはやむを得ないが, プレスキャン以外にも処理時間が伸びていることが分かる. そこで WebM および H.264/AVC の perf によるキャッシュ参照回数, キャッシュミス回数の測定を行った. その結果, WebM の場合ループスキューイング/ループインターチェンジによりラストレベルキャッシュ (以下 LLC) 参照回数が約 17 パーセント増加し, LLC ミス回数が約 2.3 パーセント増加した. 同様に, H.264/AVC の場合ループスキューイング/ループインターチェンジによ

表 12 H.264/AVC の HA8000/TS20 における 1 コア時の実行時間 [秒]

	(prescan)	予測処理	デコード全体
original	なし	30.99	46.44
loop-skewing 適用	4.99	36.75	50.44
loop-interchange 適用	4.99	42.09	55.67

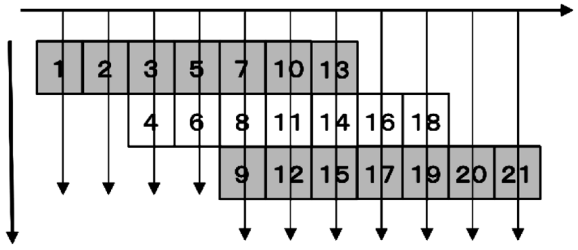

図 10 各コアのデコード処理順序 (2 コア実行時)

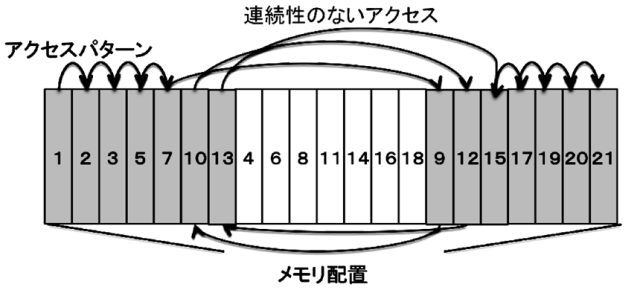

図 11 各コアのデータ列へのアクセス順序 (2 コア実行時)

り LLC 参照回数が約 27 パーセント増加し, LLC ミス回数が約 2.9 パーセント増加する様子が確認できた. LLC 参照回数の増加は L2 キャッシュミス回数の増加を表し, LLC ミス回数の増加はメインメモリへのアクセス回数の増加を示すため, 処理時間が増加したと考えられる. そこで LLC 参照回数および LLC ミス回数の差の原因を特定するために, H.264/AVC においてオリジナル版, ループスキューイングのみ適用したもの, さらにループインターチェンジを適用したものの 3 つについて 1 コア時の実行時間の調査を行った. 結果を表 12 に示す.

この結果から, ループインターチェンジの適用によってデコード時間が 5.34 秒増加し, 性能低下の原因となっていることが分かる. これは, データ列の順番通りにマクロブロックの処理が行われなくなり, 連続性のないランダムアクセスになってしまうためである. この様子を図 10 と図 11 に示す. ループインターチェンジは並列性の抽出には有効な手法であるが, 今回のケースではランダムアクセスを引き起こし性能低下に繋がってしまった.

一方, wave-front 手法による並列化では行毎に処理を完了させていくため, ループインターチェンジの際に生じた連続性のないランダムアクセスを避けることができる. これにより, オリジナル版と wave-front 手法を適用したものを比較した場合にもほとんど性能低下が見られず, 高い並列性を抽出することができたと考えられる.

5. まとめ

本稿では, WebM デコーダおよび H.264/AVC デコーダを対象とし, ループスキューイング/ループインターチェンジ手法を適用した場合と wave-front 手法を適用した場合の性能比較を Nexus7 と HA8000/TS20 上で行った.

その結果, Nexus7 上で WebM デコーダをループスキューイング/ループインターチェンジ手法で並列化した場合, 並列化箇所のみで逐次実行に比べ 3 コアで 1.26 倍の速度向上, wave-front 手法では 3 コアで 2.86 倍の速度向上が得られ, H.264/AVC デコーダをループスキューイング/ループインターチェンジ手法で並列化した場合, 並列化箇所のみで逐次実行に比べ 3 コアで 1.47 倍の速度向上, wave-front 手法では 3 コアで 1.94 倍の速度向上が得られた.

HA8000/TS20 で評価を行った結果, WebM デコーダをループスキューイング/ループインターチェンジ手法で並列化した場合, 並列化箇所のみで逐次実行に比べ 6 コアで 1.73 倍の速度向上, wave-front 手法では 6 コアで 4.61 倍の速度向上が得られ, H.264/AVC デコーダをループスキューイング/ループインターチェンジ手法で並列化した場合, 並列化箇所のみで逐次実行に比べ 6 コアで 2.18 倍の速度向上, wave-front 手法では 6 コアで 2.60 倍の速度向上が得られることが分かった.

以上より動画像デコーディングには wave-front 手法による並列処理が適していることが確認できた.

今回は OpenMP を用いて手動で並列化を行ったが, 今後は自動並列化コンパイラによる自動的な wave-front 処理の適用を行う予定である.

参考文献

[1] Z.Zhao and P.Liang: A highly efficient parallel algorithm for h.264 video encoder (2006).
[2] 中田育男：コンパイラの構成と最適化，朝倉書店 (2006).
[3] Wolfe, M.: Loop Skewing: The Wavefront Method Revisited, *International Journal or Parallel Programming* (1987).
[4] Google: The WebM Project.
[5] 大久保榮：H.264/AVC 教科書，impress (2004).
[6] OpenMP: OpenMP Website.
[7] Intel: Intel Website.
[8] ARM Corporation: ARM The Architecture for the Digital World.
[9] 渡辺裕：MPEG2 の標準化動向 (1994).
[10] 見神広紀：A highly efficient parallel algorithm for h.264 video encoder (2009).
[11] 笠原博徳：並列処理技術，コロナ社 (1991).
[12] Takahashi, M.: H.264/AVC 動画像符号化標準 (2004).
[13] ASUSTeK Computer Inc.: Nexus7 Specifications.
[14] Intel: Intel Xeon Processor X5670.