Vol.2010-SLDM-144 No.59
Vol.2010-EMB-16 No.59
Vol.2010-MBL-53 No.59
Vol.2010-UBI-25 No.59
2010/3/28

自動並列化技術を用いたメディア処理オフロード

石坂一久[†1] 酒井淳嗣[†1] 枝廣正人[†1]
宮本孝道[†2] 間瀬正啓[†2]
木村啓二[†2] 笠原博徳[†2]

自動並列化技術と自動オフロード技術により，アプリプログラムを変更することなくメディア処理をヘテロマルチコア上で高速化する手法を提案する．メディア処理は高い演算性能を必要とするが，プロセッサの進歩はヘテロマルチコアへと進んでおりソフトウェアが複雑化する．本稿では，メディア処理アプリの特徴を利用し，コアへの処理の分割と並列化をアプリプログラマに隠蔽した高速化手法を提案する．実験では，提案手法により Window Media Player のソースコードを変更することなくデコード処理をオフロードすることができ，3 コアのアクセラレータを利用して 1.8 倍の性能向上が確認できた．

Multi Media Offload with Automatic Parallelization

Kazuhisa Ishizaka,[†1] Junji Sakai,[†1]
Masato Edahiro,[†1] Takamichi Miyamoto,[†2]
Masayoshi Mase,[†2] Keiji Kimura[†2]
and Hironori Kasahara[†2]

This paper proposes new software architecture for media processing using the automatic parallelization and offload for hetero multicore. While media processing demands more and more computational power, it is difficult for a non-expert programmer to exploit hetero multicore that is a promising future processor architecture. The proposed method keeps programers away from parallelization and offloading for hetero multicore. Our experiments showed that the Windows Media Player speeduped 1.8 times by offloading the decoder to three cores accelerator without source code modification.

†1 NEC システム IP コア研究所 / System IP Core Research Lab., NEC Corporation
†2 早稲田大学 / Waseda University

1. はじめに

携帯電話やテレビなどの我々の身の回りの情報家電[⋆1]では，音声や動画像のコーデック，画像認識などのメディア処理が行われている．これらのメディア処理は，機器の主要な機能を実現している一方で，高い演算性能を必要とし，その高速化は機器開発の重要な課題である．今後も高画質化や AR 等のより高度な機能の実現など，メディア処理には高い演算性能が求められる．

プロセッサの性能向上はメニコア化，ヘテロコア化によって進みつつある．高い並列性を持つ処理は，シンプルなコアを多数持ったメニコアで処理した方が性能や消費電力の観点から有利であり，OS や並列処理に適さない処理をサポートする高性能コアとの組み合わせが有効である．本稿では前者をアクセラレータ，後者をホストと呼ぶ． メニ・ヘテロコアではコア数の増加とともに，両コアが直接共有メモリにアクセスすることや，ハードウェアでデータコヒーレンシーを保証する分散キャッシュを用いることは，性能のボトルネックになる．したがって，両コアがローカルメモリを持ちソフトウェアで明示的なデータ転送を行う方式が用いられる．本稿では，このような構成のハードウェアを対象としたメディア処理の高速化を考える（図 1 中）．特定のメディア処理の高速化を考えた場合には，アクセラレータとして専用のコアを用いる場合も考えられるが，異なるメディア処理への対応のし易さから，アクセラレータにはプログラマビリティがあるコアを想定する．また，少なくともホストには OS が搭載されるような利用形態を想定している．

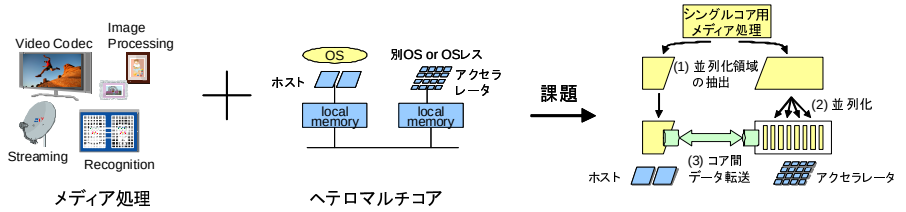

図 1 ヘテロコア上でのメディア処理高速化の課題

⋆1 情報家電とは、携帯電話、携帯情報端末（ＰＤＡ）、テレビ、自動車等生活の様々なシーンにおいて活用される情報通信機器及び家庭電化製品等であって、それらがネットワークや相互に接続されたものを広く指す.[1)]

Vol.2010-SLDM-144 No.59
Vol.2010-EMB-16 No.59
Vol.2010-MBL-53 No.59
Vol.2010-UBI-25 No.59
2010/3/28

このようなヘテロマルチコアを利用して高速化を行うためには，ソフトウェアの対応が必要となる．まず処理全体の内どこをホストで実行し，どこをアクセラレータで並列化するかを決める **(1) 並列化領域の切り出し**が必要となる．また，アクセラレータに割り当てられた処理をどのようにして並列化するかという **(2) 並列化方法**が必要である．そして，アクセラレータとホスト間のソフトウェアによる **(3) データ転送**が必要である（図 1 右）．

これらを行うには並列処理の知識や，ハードウェアアーキテクチャの理解が必要となる．しかしながら，並列プログラミングが十分に普及していない現状では，製品サイクルの短い情報家電で短期間でのソフト開発が求められる中で，一般的なソフト開発者がこれらの課題を解決して高性能なプログラムを開発することは困難である．

これらの課題を軽減するため，アクセラレータ用にチューニングされたライブラリを利用できる場合がある．例えば，GPU や DSP では数値計算や画像処理の基本的な処理を実装したライブラリが提供されている．高速化対象のメディア処理にこのようなライブラリが利用可能な場合は，アクセラレータ向けの最適化の負担を軽減できるが，ライブラリ API の学習やソースコードの変更が必要であり，利用は必ずしも容易ではない．特に，組み込み機器は既存のソフトウェア資産をベースに開発が行われるため，長年の積み重ねにより既存資産は複雑化・不透明化しており，ライブラリの利用が困難である．本稿では，ヘテロマルチコアを利用するための既存ソフトの変更を抑え，ソフト開発者の負担を軽減しながらもメディア処理を高速化する手法を提案する．

2. 提 案 手 法

本節では前述した 3 つの課題に対する我々のアプローチとして，メディア処理に適した並列化領域の切り出し，自動並列化コンパイラを用いた並列化，プログラム開発者に透過的なデータ転送について述べる．

2.1 並列化領域の切り出し

メディア処理は大きく二つの処理に分けることができる．一つ目は，コーデック処理，画像認識や音声処理などのメディア処理の中心部分であり，二つ目は，UI やファイルやネットワークの処理を行う部分である．本稿では前者をコア部，後者をインターフェース部と呼ぶことにする（図 2）．例えばビデオプレイヤーでは，画面の表示やボタン操作がインターフェース部であり，ビデオデータの復号処理がコア部である．

提案手法では，コア部とインターフェース部を切り分け，コア部のみを並列化する方法をとる．これには三つの理由がある．一つ目は，メディア処理アプリの処理時間の大部分はコア部によって占められるため，コア部の高速化がメディア処理全体の高速化への寄与が大きいからである．例えば，筆者の PC 上で行った計測では，ビデオプレイヤーの実行時間のうちデコード処理が 88%を占めていた．

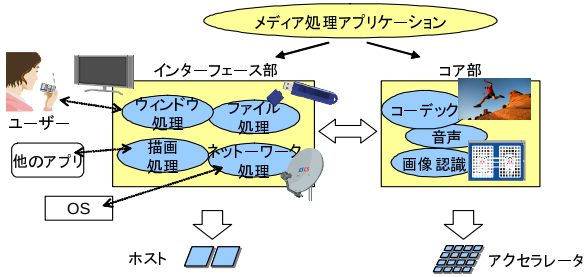

図 2 メディア処理の構成と並列化領域の切り出し

二つ目は，インターフェース部は OS を呼び出しや，必ずしも並列化されない他のアプリとの連携を行うことが多いため，インターフェース部はホストで実行したほうが効率がよいからである．

三つ目は，コア部とインターフェース部は処理内容の独立性が高いため，それぞれ異なる開発者によって開発され，コア部はライブラリとして提供されることを想定しているからである．すなわち，コア部を並列化してもライブラリの API を維持する限りは，インターフェース部変更する必要がない．したがって，インターフェース部の開発者は並列化を意識する必要がなく，見た目のカスタマイズや他のアプリとの連携などインターフェース部の本来の目的に専念することができる．このように，コア部を並列化の対象とすることで，インターフェース部のプログラム開発の負担増加を避けられるという利点がある．また後述するようにデータ転送が容易になるという利点もある．

2.2 並列化方法

次に，アクセラレータ上で実行するコア部に対する並列化の方法としては，自動並列化コンパイラを積極的に用いる方針をとる．これは，ライブラリ開発者が並列プログラミングを行うハードルを下げて開発を容易にしたいということに加え，次のような特徴を活用できる見込みがあるからである．すなわち，メディア処理の UI 部などは元来並列性が少ない上に，UI 独自の言語やツールを利用して記述されるため自動並列化に適さないが，メディア処理のコア部は高いデータ並列性を持ち，処理やデータの流れが比較的明確であることが多く，自動並列化によって高速化が期待できるからである（図 3）．最先端の自動並列化コンパイ

Vol.2010-SLDM-144 No.59
Vol.2010-EMB-16 No.59
Vol.2010-MBL-53 No.59
Vol.2010-UBI-25 No.59
2010/3/28

ラでは，MPEG2 デコードなどのメディア処理コア部に対して有効に機能し，並列化によって高速化が出きる結果が示されている[2]．

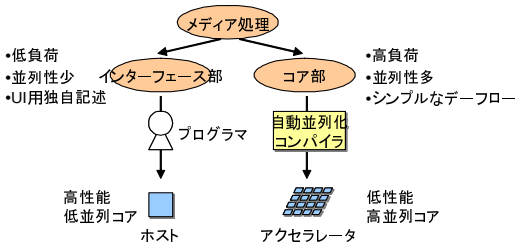

図 3 メディア処理の各部に適したプログラム開発

2.3 データ転送

提案手法では，アクセラレータで実行されるのはライブラリとして実装されたメディア処理コア部である．したがって，ホストとアクセラレータ間で流れるデータは，ライブラリとの間の入出力データとなる．提案手法では，このことを利用してアプリ開発者がデータ転送を記述する負担を削減する．

ライブラリは一般に再利用性を高めるため，公開する API を定めて内部構造を隠蔽し，内部構造を変更してもライブラリを利用するプログラム側の変更が必要ないように設計される．このように設計されたライブラリでは，ライブラリ内部のデータ構造を直接外部から触ることできず，API で定められた外部に公開された関数（以下公開関数）を通してのみ，ライブラリと外部の間でデータが受け渡しされる（図 4）．したがって，ライブラリ全体をオフロードすると，ホストとアクセラレータ間で必要なデータ転送は公開関数を通して受け渡しされるデータのみに限定できる．したがって，図 5 に示すように，データ転送を行う二つのスタブを公開関数の前に挿入する．ホスト側のスタブのインターフェースはライブラリと同一にし，アクセラレータ側のスタブはライブラリを公開関数を通して呼び出す様にすることで，呼び出し側プログラム，ライブラリの両者に対してデータ転送を隠蔽することができる．

このように，提案手法では，メディア処理コア部がライブラリとして実装されるという特徴と，ライブラリは入出力データが明確化されるという特徴を利用することで，データ転送のためにインターフェース部，コア部のプログラムの変更を不要とする．なお，ライブラリ

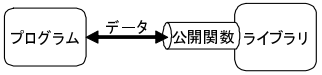

図 4 ライブラリと外部のデータフロー

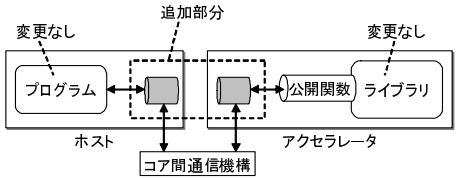

図 5 プログラムとライブラリにデータ転送を隠蔽したオフロード

全体をオフロードすると，ライブラリ中の処理時間が短い関数や並列化に向かない関数をオフロードした際のオーバヘッドが懸念されるが，今回は処理時間が長く並列化に向くと想定されるメディア処理のコア部を実装したライブラリを対象としているため，オーバヘッドを上回る高速化効果を得ることが期待できる．

なお，インターフェース部からスタブの呼び出し方法としては，スタブを直接インターフェース部にリンクする方法の他に，ダイナミックリンクの場合は実行時にライブラリコールをトラップしてスタブを呼び出すこともできる．後者の場合は，コア部をアクセラレータで実行するかホストで実行するかを容易に切り替えられるという利点がある．

2.4 提案手法のまとめ

ヘテロマルチコア上でメディア処理の高速化を行う際の 3 つの課題に対する提案手法のアプローチを表 1 にまとめる．提案手法では，メディア処理をその処理内容の特徴から，インターフェース部とコア部に分割し，処理時間負荷の大きいコア部を並列化対象とする．次に，コア部はデータや処理の流れが解析しやすいデータ並列性を持つという特性から自動並列化コンパイラによる並列化を行う．また，コア部がライブラリとして実装されることを利用し，インターフェース部とコア部の間に挿入するスタブでデータ転送を行うことで，ホストとアクセラレータ間のデータ転送を隠蔽する．

3. 実　　装

提案手法の実証のため，メディア処理としてビデオプレイヤーを例として実装を行った．実証には，NEC エレクトロニクス社製のシステム LSI「NaviEngine®（μPD35001）」[3] を用いた．本 LSI は CPU として ARM11 を 4 コア搭載した MPCore（400MHz）を備えており，今回はそのうちの 1 コアをホストとして利用し，残りの 3 コアをアクセラレータとして利用した． MPCore は主メモリを共有しているが，今回はホストとアクセラレータで分割して利用することで，メモリを共有しないヘテロ構成として利用した．なお，アクセ

Vol.2010-SLDM-144 No.59
Vol.2010-EMB-16 No.59
Vol.2010-MBL-53 No.59
Vol.2010-UBI-25 No.59
2010/3/28

表 1 メディア処理高速化の課題と提案手法

課題	解決方法	狙い	
		性能	ソフト開発
並列化領域の切り出し	メディア処理コア部の並列化	並列化に適した部分を対象とすることで効率的な高速化	インタフェース部は並列化を意識しない開発が可能
並列化方法	自動並列化コンパイラを利用	最先端の自動並列化技術で高速化	コア部のプログラマによる並列化不要
データ転送	メディア処理コア部ライブラリのオフロード	-	データ転送のためのプログラム修正が不要

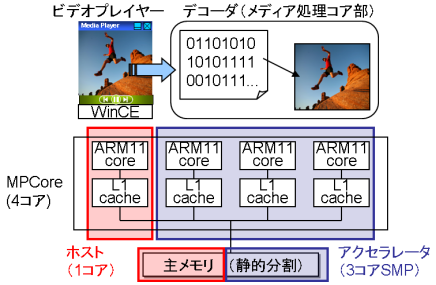

図 6 MPCore 上での並列ビデオプレイヤーの構成

ラレータを構成する 3 コア間ではハードウェアによるキャッシュコヒーレンス制御を有効とし，アクセラレータは SMP として利用した．

ホストの OS にはシングルコア用の WindowsCE（以下 WinCE）を搭載した．ビデオプレイヤーには Microsoft の Windows Media Player（以下 WMP）を用いた．標準的なアプリケーションに提案手法が適用可能であることを実証する．ビデオの符号化方式には MPEG2 を用いた．図 6 に MPCore 上での今回の実証環境の構成を示す．

3.1 メディア処理コア部

WMP はビデオデータのデコードに DirectShow を利用しており，DirectShow はコーデック毎にフィルタと呼ばれるプログラムモジュールを作成して組み込むことができる仕組みを持っている．今回は MPEG2 デコーダライブラリを呼び出すデコードフィルタを作成して組み込んだ．図 7 にビデオプレイヤーの構造を示す．この様にデコード処理とその他の処理が独立しており，前者を 2.1 節で述べたメディア処理のコア部，後者をインターフェース部と定義することができる．今回はコア部であるデコード処理が並列化対象領域である．

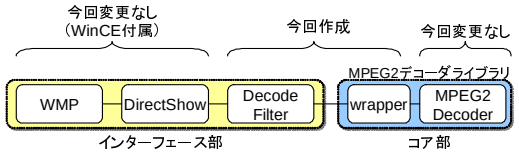

図 7 ビデオプレイヤーの構造

MPEG2 デコーダは，MediaBench[4] をベースとして，早稲田大学で開発が行われている自動並列化コンパイラ OSCAR の評価用に実装したものを用いた．今回はこれに内部構造を隠蔽するようにラッパーを追加しライブラリとした．デコーダ自体は今回の実証のための変更はしていない．

ライブラリの API は，初期化などを含めて全部で 5 つの公開関数とした．全ての関数は図 8 に示すように，引数で関数への入力データおよび出力データのアドレスとサイズを示す形式とし，ライブラリと外部のデータ入出力が明確となるようにした．また，これらの関数以外でのライブラリと外部のデータの受け渡しはなく，内部構造は隠蔽されている．図 8 は，ライブラリの主要機能で，一フレーム分の MPEG2 デコードを入力として受け取りデコードし，YUV データを出力するデコード関数である．引数の in は MPEG2 データ，out は YUV データに対応する．

```
void decoder_decode(void *in, size_t size_in,
                    void *out, size_t size_out);
```

図 8 MPEG2 デコーダライブラリのデコード関数の API

3.2 自動並列化

自動並列化には前述の OSCAR コンパイラを用いた．OSCAR コンパイラは独自のマルチグレイン並列化技術やメモリ最適化技術を用いた並列化を行う．今回新しい並列化手法は用いておらず，MPEG2 デコーダの並列化に関しては文献 2) に詳しいため本稿では述べないが，2 プロセッサで 1.9 倍，4 プロセッサで 3.2 倍等の高い性能向上が得られることが報告されている．

OSCAR コンパイラは，OSCAR API[5] を用いた並列化コードを生成する．OSCAR API は，リアルタイム組み込み低電力マルチコア向けに並列化プログラムを記述するための API である．アクセラレータでは OSCAR コンパイラが生成したコードのみを動かすため，そ

Vol.2010-SLDM-144 No.59
Vol.2010-EMB-16 No.59
Vol.2010-MBL-53 No.59
Vol.2010-UBI-25 No.59
2010/3/28

のための必要最小限のランタイムがあれば良いが，今回は実験環境の開発効率などを考慮しアクセラレータには Linux を載せ，pthread 環境を構築し，OSCAR API で記述された並列化コードを pthread コードに変換して実行させた．なお，WinCE と Linux の DualOS 環境は，MPCore 上での Toppers と Linux の DualOS 環境をベースに構築した[6]．

3.3 データ転送

ホストとアクセラレータ間のデータ転送は，2.3 節で述べたように，ライブラリとその呼び出し側との間に挿入されたスタブによって行う．ホスト側のスタブは，MPEG2 デコーダライブラリと同じインターフェースとして実装し，MPEG2 デコーダの代わりに，デコードフィルタによって呼び出される．図 8 に示したデコード関数に対するスタブの例を図 9 に示す．図中の send/recv はコア間通信機構を用いたデータ転送を表す．この例が示すように，ホスト側のスタブはまずアクセラレータにライブラリ関数の実行要求を送った後に，引数で受け取った入力データを送信する．その後出力データを受信し引数で指定されたアドレスに書き込む．このように，転送するデータはライブラリ関数の引数のみによって決定されるため，スタブはライブラリの内部構造に依存しない．

```
void host_stub_decode(void *in, size_t size_in,
                      void *out, size_t size_out) {
    struct reqest r = {DECODE, size_in, size_out};
    send(&r, sizeof(struct request));
    send(in, size_in);
    recv(out, size_out);
}
```

図 9 デコード関数のホスト側スタブ（擬似コード）

一方，アクセラレータ側のスタブを図 10 に示す．アクセラレータ側のスタブはサーバーとして実装してあり MPEG2 デコーダライブラリをリンクしている．スタブはライブラリ関数の実行要求を受け取ると，データを受信してそのデータを引数として，要求があった関数を呼び出す．関数の実行が終了すると，ライブラリ関数の引数で示された出力データを送信する．このように，転送するデータはライブラリ関数の引数のみによって決定されるため，スタブはライブラリの内部構造に依存しない．

上記スタブコード中の send, recv はホストとアクセラレータ間の通信を行うもので，図 5 のコア間通信機構に相当する。今回は主メモリ中に OS 間の共有メモリ領域とコア間割り込みを利用した通信機構を作成した．通信機構は両 OS 上のデバイスドライバとして実装した．

```
void acc_stub(void) {
    struct request r;
    recv(&r, sizeof(struct request));
    recv(in, r.size_in);
    switch (r.func_id) {
        case DECODE:
            decoder_decode(in, r.size_in, out, r.size_out);
                :
    }
    send(out, r.size_out);
}
```

図 10 アクセラレータ側スタブ（擬似コード）

3.4 並列化のためのソフト開発負担削減

今回作成した並列ビデオプレイヤーの構成を図 11 に示す．灰色部分は並列化を行わない場合（図 7）と変更がないことを示す．本実装を通して，ソフト開発者の並列化のための負担を軽減するという提案手法の狙いが，以下のように達成できていることが確認できた．

- WMP を変更していない → インターフェース部を並列化を意識せずに開発可能
- MPEG2 デコーダを変更していない → コア部の開発者自身による並列化が不要

なお，今回は少数のスタブを手動で作成して実験したが，今回の MPEG2 ライブラリのように公開関数の引数から機械的に転送データが決定できる場合には，スタブの自動生成を行うことで，よりプログラマの負担を軽減できると考えている．

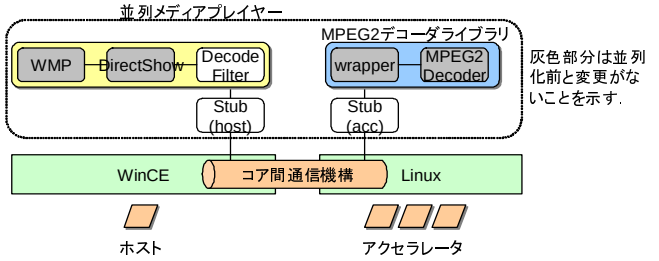

図 11 並列メディアプレイヤーの構成

Vol.2010-SLDM-144 No.59
Vol.2010-EMB-16 No.59
Vol.2010-MBL-53 No.59
Vol.2010-UBI-25 No.59
2010/3/28

4. 高速化評価

提案した並列メディアプレーヤーで MPEG2 再生を行った測定結果を述べる．MPEG2 を再生するときに 1 フレームを処理するのに要する時間を図 12 に示す．図はホスト単体のみで再生した場合（図中コアホストのみ），提案手法を用いてデコード処理をアクセラレータにオフロードし並列処理した場合（図中提案手法）の場合について，ホスト単体の場合の実行時間で正規化して示している．図の提案手法の場合でホストとアクセラレータのバーが重なっているが，描画処理とデコード処理がオーバラップして実行されることを表している．

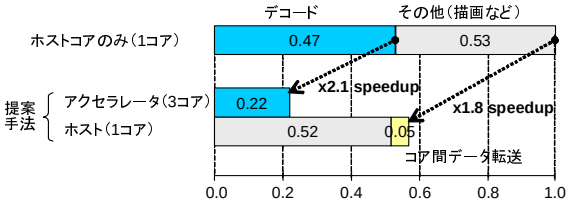

図 12 ビデオプレイヤーでの 1 フレームの処理に用する時間の内訳

まず，デコード部分のみの並列化効果を見ると，1 コアの場合に比べて 2.1 倍の速度向上が得られ，コア部に対する自動並列化による速度向上が確認できた．また，今回は合計 4 コアまでの評価であるが，文献 7) では 8 コアの IBM p5 550Q SMP サーバー上で 4 コアで 3.46 倍, 8 コアで 5.12 倍という速度向上が報告されており，スケーラビリティも期待できる．メディア処理全体としては 1.8 倍の性能向上が得られた．これにはコア部の並列化の他に，コア部とインターフェース部が並列実行されることも寄与している．なお，今回は描画処理の最適化を行っていないため，全体としての高速化を抑えているが，描画処理の最適化で更なる高速化が期待できる．

ところで，ワンチップ上でメディアプレーヤーとデコーダの両方を動かすと性能干渉が気になるが，今回と同一環境（ただし MPCore 上の 4 コアを SMP として利用）で，OSCAR コンパイラで並列化した MPEG2 デコーダを 3 コアで動かし，残りの 1 コアで画面表示プログラムを動かしたときの性能の干渉が無視できる程度であることが報告されている[8]．

ビデオプレイヤーでの提案手法の実装方法とその効果を表 2 にまとめる．

表 2 ビデオプレイヤーでの提案手法の実装とその効果

課題	提案手法の実装	効果	
		性能	ソフト開発
並列化領域の切り出し	デコード処理をコア部として切り出す	オーバヘッドを上回る並列化効果で 1.8 倍の高速化	プレイヤーのインターフェース部（WMP）の変更なし
並列化方法	OSCAR コンパイラで自動並列化	デコーダのみでは 3 コアで 2.1 倍の高速化	MPEG2 デコーダの手動並列化なし
データ転送	MPEG2 デコーダライブラリ全体のオフロード	-	WMP，MPEG2 デコーダとも変更なし

5. 関 連 技 術

自動並列化技術は，本文中でも述べたようにメディア処理や科学技術計算で有効であることが示されている．しかし，筆者らの知る限りメディアプレイヤーのような実用的なアプリ全体に適用された例はなく，本稿で述べたようにアプリの一部に用いる方法が現時点では現実的である．提案手法では，メディア処理が並列化に適したコア部とインターフェース部に分かれることを利用して自動並列化コンパイラを有効に活用する．

並列プロミングとしては，共有メモリ型マルチコア用にはディレクティブベースの言語拡張である OpenMP やライブラリベースの Intel の TBB などがあるが，分散メモリはサポートしておらず，本稿で対象とする様なヘテロコア向けの記述はできない．一方，分散メモリ用の並列言語として分散コンピューティングなどで利用される MPI，GPGPU 向けの CUDA，OpenCL などがあるが，これらではデータ転送は明示的に記述する必要があり，一般的なプログラマには敷居が高いことは，OpenCL が熟練プログラマを対象していることからも伺われる[9]．また，RPC では IDL やソースコード中のアノテーションからデータ転送コードを自動生成する方法が取られるが，転送可能なデータに制約があり，一般には IDL やアノテーションの追加だけでなく，プログラムを専用に設計する必要がある．これらの言語拡張やフレームワークでは，それらの学習やプログラム変更が必要なことも課題となる．提案手法では，ライブラリは外部とのデータフローが明確になるように設計されることを利用してデータ転送コードを生成することで，プログラムの変更が必要ない．なお，今回の実験では独自のコア間通信機構を利用したが，データ転送の記述方法には依存しないため，言語や RPC で提供される記述方法を利用することも可能でる．

CPU での TCP プロトコル処理をオフロードするために TCP オフロードエンジンが利用される．OS のプロトコル処理をエンジンへオフロードするため，システムコールで OS

Vol.2010-SLDM-144 No.59
Vol.2010-EMB-16 No.59
Vol.2010-MBL-53 No.59
Vol.2010-UBI-25 No.59
2010/3/28

のプロトコル処理を利用しているアプリは，プログラムを変更することなく高速化される．しかし，メディア処理は TCP プロトコル処理と異なり通常ユーザーレベルで実行されるため，OS によってオフロード処理を隠蔽することはできない．

アプリの一部の処理をオフロードする方式としてプラグインを利用する方法も考えられる．プラグインはアプリケーションが定める API に従って記述されたプログラムで，アプリの機能を拡張するものであり，Web ブラウザの Flash プラグインなどが代表的である．プラグインとアプリ間のデータ転送を API で明確に定義することで提案手法を応用することができ，プラグインを利用するアプリをヘテロコア上で高速化することが可能である．

6. おわりに

本稿では，ヘテロコア上でメディア処理を高速化する手法を提案した．プロセッサのメニコア・ヘテロコア化が進み，ソフトウェアの並列化や明示的なデータ転送が必要などソフトウェアが複雑化している．本稿では，メディア処理の特徴を利用しすることで，これらをプログラマに隠蔽し，プログラムを変更することなくヘテロコアを活用する方法を提案した．

組み込み機器では，多数のアプリが複雑に連携しているシングルコア用のソフト資産のマルチコアへの移行が課題となっている．既存部分の変更を最小限に抑えながらメディア処理のオフロード，高速化を実現する提案手法は，ヘテロコア上で既存資産を活用しながらより高機能なメディア処理を組み込むことを可能とし，競争力の高い製品の開発に貢献する．

参考文献

1) 経済産業省. 基本戦略報告書「e-life イニシアティブ」, Apr. H15. http://www.meti.go.jp/kohosys/press/0003917/.
2) 宮本他. 情報家電用マルチコア上におけるマルチメディア処理のコンパイラによる並列化. SACSIS2008 - 先進的計算基盤システムシンポジウム, May 2008.
3) NaviEngine®. http://www.necel.com/automotive/ja/assp/naviengine.html.
4) C. Lee et al. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems. *30th International Symposium on Microarchitecture(MICRO-30)*, 1997.
5) Keiji Kimura, Masayoshi Mase, Hiroki Mikami, Takamichi Miyamoto, Jun Shirako, and Hironori Kasahara. OSCAR API for Real-time Low-Power Multicores and Its Performance on Multicores and SMP Servers. *Proc. of The 22nd International Workshop on Languages and Compilers for Parallel Computing (LCPC2009)*, Oct. 2009.
6) 阿部, 酒井. 組込み向けマルチコアプロセッサ MPCore を用いた応答性/機能性両立環境評価－制御処理と情報処理の融合にむけて－. 信学技報 [CPSY2007-83,DC2007-87], Vol. 107, No. 558, pp. 19–24, May 2008.
7) 宮本他. マルチコア上でのマルチメディアアプリケーションの自動並列化. 情報処理学会研究会報告 2007-ARC-171-13, Jan. 2007.
8) 宮本他. 組み込みマルチコア上での複数アプリケーション動作時の自動並列化されたアプリケーションの処理性能. 情報処理学会研究報告, Mar. 2010.
9) Khronos OpenCL Working Group. The OpenCL Specification Version: 1.0 Document Revision: 29. Dec 2008.