Vol.2010-ARC-190 No.7
2010/8/3

情報家電用ヘテロジニアスマルチコア用自動並列化コンパイラフレームワーク

林 明宏[†1] 和田康孝[†1] 渡辺岳志[†1]
関口威[†1] 間瀬正啓[†1] 木村啓二[†1]
伊藤雅之[†2] 長谷川淳[†2] 佐藤真琴[†3]
野尻徹[†3] 内山邦男[†3] 笠原博徳[†1]

汎用 CPU コアに加え特定処理を高効率で実行可能なアクセラレータを搭載したヘテロジニアスマルチコアが広く普及している．しかしながら，ヘテロジニアスマルチコアでは様々な計算資源へのタスクスケジューリングやデータ転送コード挿入等多くの負担をプログラマが負う必要がある等プログラミングが困難である．そこで本稿では，複数 CPU 及びアクセラレータを持つヘテロジニアスマルチコアに対して，逐次プログラムを入力とし自動的に実行効率の良い並列プログラムを生成する，ヘテロジニアスマルチコア向け自動並列化コンパイラフレームワークを提案する．本フレームワークでは自動並列化コンパイラとアクセラレータコンパイラとのインターフェースとして新たに提案するヘテロジニアスマルチコア向け OSCAR API を利用することで，逐次 C プログラムを自動的に汎用コアとアクセラレータコアにタスクを配分し，高い性能を実現する．本手法を情報家電用ヘテロジニアスマルチコアプロセッサ RP-X をターゲットとして，AAC エンコーダ及び Optical Flow 計算の自動並列化性能を評価した．その結果，8 つの汎用 CPU コア及び 4 つのアクセラレータコアを使用した場合，逐次実行時と比較して Optical Flow 計算で約 12 倍（OSCAR コンパイラ+アクセラレータコンパイラ使用時），約 32 倍（OSCAR コンパイラ+既存ライブラリ使用時），AAC エンコーダで約 16 倍（OSCAR コンパイラ+既存ライブラリ使用時）の性能向上が得られ，ヘテロジニアスマルチコアを対象とした汎用的なコンパイラフレームワークを実現可能であることがわかった．

A Compiler Framework for Heterogeneous Multicores for Consumer Electronics

Akihiro Hayashi,[†1] Yasutaka Wada,[†1]
Takeshi Watanabe,[†1] Takeshi Sekiguchi,[†1]
Masayoshi Mase,[†1] Keiji Kimura,[†1] Masayuki Ito,[†2]
Atsushi Hasegawa,[†2] Makoto Sato,[†3] Tohru Nojiri,[†3]
Kunio Uchiyama[†3] and Hironori Kasahara [†1]

Heterogeneous multicores, which integrates multiple general purpose CPU cores and special purpose accelerator cores on a chip, has been widely used in order to attain high performance keeping power consumption low. However, heterogeneous multicores require to programmers very difficult coding for load distribution to CPU cores and accelerator cores, synchronizations and data transfer using DMA controllers. To this end, this paper proposes a compiler framework which facilitates the development of the program for heterogeneous multicores. This framework parallelize the sequenctial C program using OSCAR parallelizing compiler and accelerator compiler. The developed framework gives us 12 times, 32 times and 16 times speedup with eight general purpose CPU cores and four accelerator cores on RP-X processor for an Optical Flow Calculation(using accelerator compiler), Optical Flow Calculation(using library) and an AAC audio encoder program(using library), respectively, against sequential execution by a single CPU core.

1. はじめに

マルチコアプロセッサに加え特定処理を高い性能で実行可能なアクセラレータを搭載したヘテロジニアスマルチコアが注目を集めている．その理由は，低消費電力で高い性能を実現可能なその特性にあり，これまで，多くのヘテロジニアスマルチコア及びそのソフトウェアがハイパフォーマンスコンピューティング分野や組み込み分野，デジタル民生機器等，適

†1 早稲田大学
Waseda University
†2 ルネサスエレクトロニクス株式会社
Renesas Electronics Corporation.
†3 株式会社日立製作所
Hitachi, Ltd.

用分野を問わず提案あるいは製品化されている．例えば従来までに CELL[1)]，GPGPU[2)]，Larrabee[3)]，RP1[4)]，RP-X[5)]，NaviEngine[6)]，UniPhier[7)]，CUBA[8)] 等が開発，市販されている．

しかしながら，ヘテロジニアスマルチコアでは異種の計算資源へのタスクスケジューリング，同期コードや DMA を用いたデータ転送コード挿入等多くの負担を開発者が負うことになる．このような並列プログラムを短期間で開発し，製品の市場競争力を強化するためには，並列プログラミング，ソフトウェアチューニングの困難さを緩和する必要がある．

この問題を解決するため，NVIDIA は CUDA[9)]，AMD は CTM[10)]，Khronos Group は OpenCL[11)] を提案しており，アクセラレータのプログラミング及び，汎用コアとのインターフェースを API で抽象化することでヘテロジニアスマルチコア用の並列化プログラミングを容易にした．しかしこれらは，アクセラレータプログラミングの容易化に焦点をあてており，汎用コアとの負荷分散，メモリローカリティの最適化等，ヘテロジニアスマルチコアシステム全体をターゲットとしていない．

これに対し C.K Luk らは，汎用 CPU の並列化プログラミングに Intel Threading Building Block(TBB)[12)]，アクセラレータプログラミングに CUDA[9)] を利用し，実行時に動的に演算資源に対してタスクを割り当て，負荷の分散を実現するフレームワーク Qilin[13)] を提案している．しかしながら，これら従来手法は依然としてプログラマが手動で並列化を行う必要があり，根本的な問題解決には至っていない．これらに対し，Bellens らは，Cell BE[1)] を対象とし，逐次プログラムにデータフロー情報を加えたプログラムを SPE に対して動的に自動負荷分散するフレームワーク CellSs[14)] の提案を行っている．しかし，計算資源を SPE のみとしており，ヘテロジニアスマルチコア向けの並列化を行ってはいない．

以上を踏まえ，本稿ではヘテロジニアスマルチコアの性能を最大限にかつ容易に引き出すために逐次プログラムを入力とし，自動的に並列化コードを生成するヘテロジニアスマルチコア API(Application Program Interface) および OSCAR コンパイラ[15)] を中心としたコンパイラフレームワークを提案する．

本稿では以下の 2 点について述べる．

- 汎用性の高いコンパイルフローの提案．
  - アクセラレータコンパイラを利用した OSCAR コンパイラによる自動並列化
  - 既存のアクセラレータライブラリを利用した OSCAR コンパイラによる自動並列化
- OSCAR API[16),17)] のヘテロジニアスマルチコア向け拡張

以下 2 章で提案フレームワークが対象とするヘテロジニアスマルチコアアーキテクチャについて，3 章でヘテロジニアスマルチコア API およびコンパイラフレームワークについて．4 章でメディアアプリケーションを用いた性能評価について述べる．

2. ヘテロジニアスマルチコア向け OSCAR API 対象アーキテクチャ概要

本章では OSCAR API の概要及び，OSCAR API が及び提案フレームワークが対象とするアーキテクチャの概要について述べる．

2.1 OSCAR API 概要

OSCAR API は NEDO 半導体アプリケーションチッププロジェクト「リアルタイム情報家電用マルチコア技術の研究開発」において，(株）日立製作所，(株）ルネサステクノロジ（当時），(株）富士通研究所，(株）東芝，松下電器産業（株）（当時），日本電気（株）および早稲田大学の構成員からなる，マルチコア・アーキテクチャ・API 検討委員会にて検討及び策定された並列化 API である．本 API の特徴は，共有メモリ・マルチプロセッサ用並列化 API の業界標準である OpenMP のサブセットを用いることにより OpenMP コンパイラで並列バイナリを生成でき，さらに情報家電用マルチコアで必要とされるメモリ配置，データ転送，電力制御，及びリアルタイム処理用にタイマ制御の指示文を定義したことにある．本 API を使うことにより，ユーザがアプリケーションを並列化することはもちろん，OSCAR 自動並列化コンパイラによる並列化や低消費電力最適化を，各社の共有メモリ並列サーバや情報家電用マルチコアに適用することが可能となる．以下で提案手法が対象とするアーキテクチャ，3 章でヘテロジニアスマルチコア向けの API 拡張について述べる．

2.2 OSCAR API Applicable ヘテロジニアスマルチコアアーキテクチャ

提案手法が対象とするアーキテクチャを図 1 に示す．本アーキテクチャは複数個の汎用コア及びアクセラレータコア，そして集中共有メモリ (CSM) から構成されるヘテロジニアスマルチコアである．各コアはバスやクロスバースイッチ等の相互接続網で接続されている．アクセラレータコア (ACC) の接続形式は，(1）汎用コアと同梱されているもの（コントローラ付きアクセラレータ），(2）バスに直接接続されているもの（コントローラなしアクセラレータ）の両方をサポートしている．

また，本アーキテクチャでは，汎用コア及び，コントローラ付きアクセラレータはローカルデータメモリ (LDM)，ローカルプログラムメモリ (LPM)，分散共有メモリ (DSM)，データ転送ユニット (DTU)，周波数/電圧制御レジスタ (FVR)，命令キャッシュメモリ，データキャッシュメモリを持っている．LDM はコアプライベートのデータを格納するためのメモリ，DSM は，自コアと他コアの双方から同時にアクセス可能なデュアルポートメモリであ

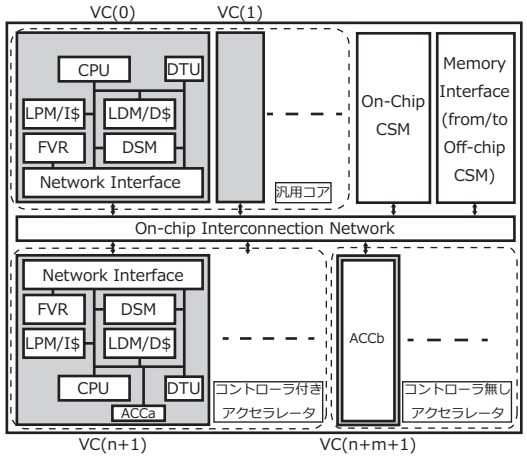

図 1 OSCAR-API Applicable ヘテロジニアスマルチコアアーキテクチャ
Fig. 1 OSCAR-API Applicable heterogeneous multicore architecture

り，タスク間データ転送や同期フラグの授受に使用される．DTU は各コアによるタスク処理とは独立にデータ転送を行うことが可能な高度 DMA コントローラであり，タスク処理とデータ転送がオーバーラップ可能とする．そして，チップ上及びチップ外に集中共有メモリ (CSM) が用意され，各コアで共有されるデータを格納する．

本アーキテクチャは，既に各社が製品化しているヘテロジニアスマルチコアのスーパーセットであるため，本アーキテクチャを API 開発のターゲットとすることで，様々なヘテロジニアスマルチコア上で提案コンパイラフレームワークが動作可能である．

3. ヘテロジニアスマルチコア向けコンパイラフレームワーク

本節では，提案コンパイルフレームワークの詳細について述べる．

3.1 提案コンパイラフレームワーク概要

本手法では，まず（1）逐次 C プログラム中のアクセラレータ実行可能箇所がアクセラレータコンパイラもしくは開発者によって付加され，その情報をもとに（2）OSCAR コンパイラによる粗粒度タスクのヘテロジニアス並列化[18] が行われ，最終的に（3）OSCAR API を使用した並列化コード生成及び実行バイナリの自動生成が行われる．なお，アクセラレータの実行バイナリとして既存のライブラリを用いる事も可能である．

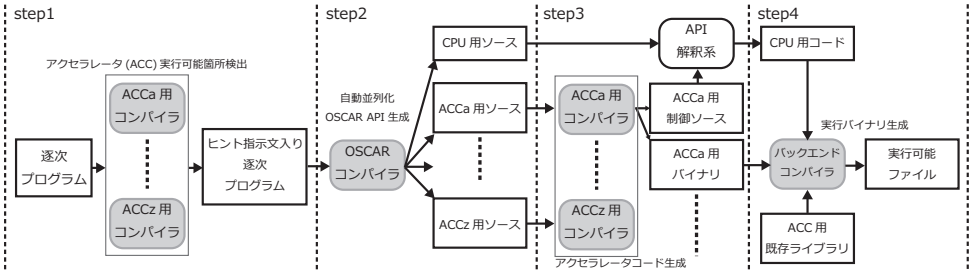

図 2 提案フレームワークのコンパイルフロー
Fig. 2 Compilation flow of the proposed framework

3.2 コンパイルフロー

図 2 にコンパイルフローを示す．本フローは，主に OSCAR ヘテロジニアス並列化コンパイラ，アクセラレータコンパイラ，バックエンドコンパイラの 3 つの処理系からなり，逐次 C プログラムを入力として，対象ヘテロジニアスマルチコア用の実行バイナリを生成する．本コンパイルフローを用いた場合の処理の流れは以下の様になる．

step 1: 逐次の C プログラムをアクセラレータ用コンパイラに通し，プログラム中のアクセラレータ実行可能部分やアクセラレータでの処理コストなどの情報を「OSCAR コンパイラ用ヒント指示文」としてソースプログラム中に挿入する．

step 2: OSCAR コンパイラは各アクセラレータ用コンパイラが挿入した OSCAR コンパイラ用ヒント指示文入りソースプログラムを入力とし，ヘテロジニアスマルチコア用並列化や低消費電力処理を行い，その結果を「OSCAR ヘテロジニアスマルチコア指示文」入り C プログラムとして出力する．この OSCAR ヘテロジニアスマルチコア指示文は OSCAR API をもとに，後述するヘテロジニアスマルチコア用の拡張を加えたものである．この時，OSCAR コンパイラはアクセラレータの種類ごとに各アクセラレータで実行されるプログラム部分を切り出し，これらを汎用 CPU 用コードとは別に出力する．

step 3: OSCAR コンパイラ出力以後は，汎用 CPU コード中の OSCAR API 指示文は API 解釈系により対象 CPU 用の実行時ライブラリに変換され，さらに，バックエンドコンパイラでオブジェクトコードに変換される．アクセラレータ用コードはそれぞれ対象とするアクセラレータコンパイラによりアクセラレータ用バイナリに変換され，同時にアクセラレータを制御するコントローラ CPU 用コードが生成される．このコント

ローラ CPU コードは，汎用 CPU 用バックエンドコンパイラによりオブジェクトコードに変換される．

step 4: 最終的に，これらの汎用 CPU 用コードとアクセラレータコードが，リンカにより単一の実行オブジェクトとしてリンクされる．

本フレームワークでは，既存のアクセラレータ用ライブラリコードを有効活用することを目的に，OSCAR コンパイラ用ヒント指示文でライブラリにより実行可能な部分を指定することができる．指定箇所が OSCAR コンパイラによりアクセラレータに割り当てられた場合は，リンク時にライブラリ中のオブジェクトをリンクされる．

3.3 OSCAR コンパイラ向けヒント情報

前項 step1 で挿入される OSCAR コンパイラ用ヒント指示文を以下に示す．(図 3)

```
#pragma oscar_hint accelerator_task (accelerator_type)
cycle (exec_cycle,[trans_mode]) [workmem(mem_type,mem_size)]
[in(in_list)] [out(out_list)] new-line

#pragma oscar_comment [ "comments" ]
```

図 3 OSCAR コンパイラ向けヒント情報
Fig. 3 Hint directive for OSCAR compiler

accelerator_task はアクセラレータによって実行可能なブロックを指定するヒント指示文である．ブロックとは，ループ，基本ブロック，関数あるいはサブルーチン呼び出しのいずれかとなる．対象ブロック内部にアクセラレータ用のライブラリ呼び出しを含むこともできる．引数として，アクセラレータの種別を accelerator_type に，アクセラレータで実行した場合の実行サイクル数（汎用 CPU におけるクロック周波数換算）を exec_cycle で，管理コアとアクセラレータ間のデータ転送方法を trans_mode で，管理コアが作業用に使用するメモリ種別と容量を mem_type と mem_size で，対象ブロックへの入力となる変数リストを in_list で，出力となる変数リストを out_list で，それぞれ指定することができる．trans_mode 及び mem_type は 3.5 項で述べるモジュール記述を用いて指定する．trans_mode が省略された場合，データ転送は CPU で行われることになる．accelerator_task により指定されたブロックは，OSCAR コンパイラのコンパイル結果により実際にどのアクセラレータで実行されるか決定される，

```
int main() {
  int i, x[N], var1 = 0;
  /* MT1 */
  for (i = 0; i < N; i++) { x[i] = i; }
  /* MT2 */
#pragma oscar_hint accelerator_task (ACCa) cycle(1000,((OSCAR_DMAC())))
  for (i = 0; i < N; i++)  { x[i]++; }
  /* MT3 */
#pragma oscar_hint accelerator_task (ACCb) cycle(100) in(var1,x[2:11]) out(x[2:11])
  call_FFT(var1, x);
  return 0;
}
```

```
void call_FFT(int var, int* x) {
#pragma oscar_comment "XXXXX"
  FFT(var, x);
}
```

図 4 OSCAR コンパイラ向けヒント情報例
Fig. 4 Example of source code with hint directives

oscar_comment は OSCAR コンパイラの出力コードに含めるべきコメント文を指定する．本指示文は，アクセラレータ用コンパイラに対してコメント文を与えるなどといった用途を想定している．本指示文は accelerator_task 指示文によって指定されたブロックの内部に記述可能である．

図 4 に OSCAR コンパイラの入力となるヒント指示文入りプログラム例を示す．このプログラムでは/* MT2 */以下のループがアクセラレータ ACCa で，/* MT3 */以下の関数呼び出しが ACCb で実行可能であるという情報が付与されている．ACCa で実行可能なループは当該アクセラレータにおいて 1000 クロックサイクルで実行可能であり，データ転送には DMAC を利用する．ACCb で実行可能なループは当該アクセラレータにおいて 100 クロックサイクルで実行可能であり，データ転送には CPU を利用し，var1 と配列 x の 2 番目の要素から 11 番目の要素が入力，配列 x の 2 番目の要素から 11 番目の要素が出力される．また，call_FFT() 関数内に OSCAR コンパイラ出力後のアクセラレータコンパイラに渡すコメントが oscar_comment により付与されている．

3.4 OSCAR ヘテロジニアス並列化コンパイラ[18)]

本節では OSCAR ヘテロジニアス並列化コンパイラについて述べる．OSCAR ヘテロジニアス並列化コンパイラは逐次プログラムを入力とし，(1) 粗粒度タスクの生成 (2) ヘテロジニアスマルチコア用の粗粒度タスクスタティックスケジューリング (3) タスク間同期コードを含めた並列化コードの生成を行う．粗粒度タスク並列処理はループ並列処理等他の並列処理方式に比べ，より大きな粒度でプログラムを並列化し，処理性能の向上に効果的である

ため，OSCAR ヘテロジニアス並列化コンパイラでは粗粒度タスク並列処理を適用する．

ヘテロジニアス並列化の第一段階として，OSCAR コンパイラはソースプログラムを基本ブロックやループやサブルーチン等の粗粒度タスク (MT) に分割する．MT 生成後，OSCAR コンパイラは MT 間のコントロールフローとデータ依存関係を表現したマクロフローグラフ (MFG) を生成し，さらに MFG から MT 間の並列性を最早実行可能条件解析により引きだした結果をマクロタスクグラフ (MTG) として表現する[19)20)]．ヘテロジニアス並列化の第二段階として，OSCAR コンパイラは MTG 中の各 MT をチップ上の計算資源の特性を考慮しスケジューリングする[18)]．スケジューリングアルゴリズムはリストスケジューリングにおける優先度及びタスク割り当ての方針をヘテロジニアスマルチコア向けに拡張したものである．最後に OSCAR コンパイラは MT のスケジューリング結果を元に OSCAR API ソースを出力する．

3.5 コア番号及びチップリソースの管理方法

ヘテロジニアスマルチコアには汎用コアの他に各種アクセラレータが同一システム上に混載されるため，フレームワーク及び API 中で，これらを統一的に扱う必要がある．このような目的から，VC (Virtual Core) 番号を定義する．VC 番号は 0 から始まり，以下のコア種別ごとに連続した番号が割り振られる．

(1) 汎用コア (図 1 中 VC(0) - VC(n))
(2) コントローラ付きアクセラレータ:(図 1 中 VC(n+1) - VC(n+m))
(3) コントローラなしアクセラレータ:((図 1 中 VC(n+m+1) -)
(4) その他（DMAC 等)

さらに，チップやコアに存在するメモリなどのモジュールを統一的に扱う記述方法を定めた．これを図 5 に示す．これらは，OSCAR API 上で対象モジュールの電力制御などに使用される． 図 5 において，chip はチップ番号，vc は VC 番号，module は OSCAR_を接

```
[[chip, ] vc,] MODULE_ARG_LIST
 MODULE_ARG_LIST := MODULE_ARG, MODULE_ARG_LIST
 MODULE_ARG := (module([sub_module])[, arg_list])
```

図 5 チップ内モジュール指定方法
Fig. 5 Specifying the module within a chip

頭辞としたモジュール名，sub_module はサブモジュール番号，arg_list は指示文でそのモジュールに対して指定する引数リストとなる．chip や vc が省略された場合は，本記述を用いた指示文が実行されるコアが指定されたものとする．また，chip もしくは vc に-1 が指定された場合は，全チップあるいはチップ内部の全コア共有のモジュールであることを表す．このような記述により，任意のチップ・任意のコアに存在するモジュールを統一的に扱うことが可能である．

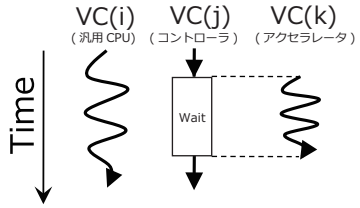

図 6 対象アーキテクチャ上での実行モデル
Fig. 6 Execution model on the architecture

3.6 アクセラレータ実行モデル

アクセラレータコア上での処理実行には一般に以下の制御コードを汎用コアが実行する必要がある．本稿ではこのような処理を行うプロセッサをコントローラと定義する．

- プログラムのロード
- アクセラレータへのデータ供給
- アクセラレータの起動
- アクセラレータによる演算結果の回収

前述のアーキテクチャにおいては，アクセラレータの接続方式によってアクセラレータ実行モデルが異なる．図 6 に対象アーキテクチャ上でのアクセラレータ実行モデルを表す．コントローラ付きアクセラレータでは，同梱されている汎用コアがコントローラとなる ($j = k$)．一方，コントローラなしアクセラレータではチップ上の汎用コアのどれかがコントローラとして実行される ($j \neq k$ となる)．また，コントローラの処理はアクセラレータの実行中はブロックされる．

3.7 ヘテロジニアスマルチコア向け OSCAR API を用いたコード生成

ヘテロジニアスマルチコア向け OSCAR API を以下に示す．

本 API は callee で指定した関数あるいはサブルーチンが VCno で指定されたコアから呼び出されることを指定する．アクセラレータ用コンパイラは，callee 中の一部あるいは全ての処理に対してアクセラレータで実行可能なオブジェクトを生成する．また，アクセラレー

```
#pragma oscar_hint accelerator_task_entry [controller (VCno)] callee, new-line
```

図 7 ヘテロジニアスマルチコア用 OSCAR API
Fig. 7 OSCAR API for heterogeneous multicore

```
int main() {
#pragma omp parallel sections
  {
#pragma omp section
    { MAIN_CPU0(); } /* 汎用 CPU スレッド */
#pragma omp section
    { MAIN_CPU1(); } /* ACCb 制御スレッド */
#pragma omp section
    { MAIN_CPU2(); } /* ACCa 制御スレッド */
  }
  return 0;
}
```

```
int MAIN_CPU1() {
  ...
  oscartask_CTRL1_call_FFT(var1, &x);
  ...
}
int MAIN_CPU2() {
  ...
  oscartask_CTRL2_call_loop2(&x);
  ...
}
```

図 8 ヘテロジニアスマルチコア API 出力例
Fig. 8 Example of parallelized source code with OSCAR API

タ用コンパイラは callee で指定された関数中にアクセラレータを起動するための管理コア用コードを生成する．

図 4 のプログラムを，2 基の CPU (VC0-VC1)，1 基のコントローラ付きアクセラレータ ACCa (VC2)，及び 1 基のコントローラなしアクセラレータ ACCb (VC3) の構成のヘテロジニアスマルチコアアーキテクチャ用にコンパイルすることを考える．ここで，VC1 がコントローラなしアクセラレータ VC3 の駆動などの制御を行う．図 8 に OSCAR コンパイラが出力する汎用コア用のコード，図 9 に VC2，VC3 用のコードをそれぞれ示す． 図 8 では，main() 関数中で汎用コア用のスレッドを MAIN_CPU0()，MAIN_CPU1()， 及び MAIN_CPU2() として parallel sections 内部でそれぞれ生成する．MAIN_CPU1() 内部では，VC3 すなわち ACCb で実行される処理を含む oscartask_CTRL1_call_FFT 関数を呼び出す．同様に，MAIN_CPU2() では VC2 すなわち ACCa で実行される処理を含む oscartask_CTRL2_loop2() 関数を呼び出す．図 9 では，accelerator_task_entry により CPU から呼び出される oscartask_CTRL2_loop2() が指定されている．アクセラレータ用コンパイラは，この情報をもとにアクセラレータ起動用のコードを生成する．さらに，関数中のアクセラレータで処理可能な部分に対するアクセラレータ用オブジェクトを生成する．同様

```
/* ACCa 用コード : MAIN_CPU2 から call */
#pragma oscar accelerator_task_entry controller(2) oscartask_CTRL2_loop2
void oscartask_CTRL2_loop2(int *x) {
  int i;
  for (i = 0; i <= 9; i += 1) { x[i]++; }
}
/* ACCb 用コード : MAIN_CPU1 から call */
#pragma oscar accelerator_task_entry controller(1) oscartask_CTRL1_call_FFT
void oscartask_CTRL1_call_FFT(int var1, int *x) {
#pragma oscar_comment "XXXXX"
  oscarlib_CTRL1_ACCEL3_FFT(var1, x);
}
```

図 9 ヘテロジニアスマルチコア API 出力例
Fig. 9 Example of parallelized source code with OSCAR API

に，accelerator_task_entry により CPU から呼び出される oscartask_CTRL1_call_FFT が指定されている．この関数中の oscarlib_CTRL2_ACCEL3_FFT() 関数は，アクセラレータ用のライブラリ関数である．た，図 4 に存在した oscar_comment がそのまま付与されていることが分かる．

4. 性能評価

本章では提案フレームワークを用いて，メディアアプリケーションの性能を情報家電用マルチコア RP-X 上評価した結果について述べる．

4.1 評価環境

本評価では，情報家電用ヘテロジニアスマルチコア RP-X[5] を用いて，提案フレームワークが高い性能を実現することを示す．RP-X は汎用コアとして 648MHz で動作する SH-4A コアを 8 基，アクセラレータコアとして 324MHz で動作する FE-GA[21] を 4 基，その他種々のハードウェア IP を搭載したヘテロジニアスマルチコアである (図 10) 各汎用コア内メモリは命令キャッシュ(32KB)，データキャッシュ(32KB)，ローカルメモリ (ILM, DLM:16KB)，分散共有メモリ (URAM:64KB)，データ転送ユニットを持つ．また，アクセラレータコアはコントーローラなしアクセラレータであり，オンチップバス (SHwy#1) に接続されている OSCAR-API Applicable アーキテクチャである．

RP-X 内部はコヒーレンシ制御を行うハードウェアを持つ 4 コアの SMP がクラスタを構成している．クラスタ間ではハードウェアによるコヒーレンシ制御を行わないため，コンパ

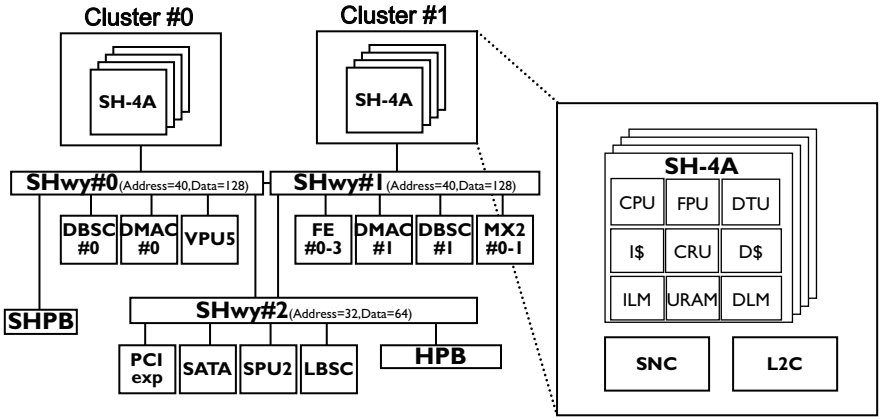

図 10 情報家電用ヘテロジニアスマルチコア RP-X
Fig. 10 RP-X heterogeneous multicore for consumer electronics

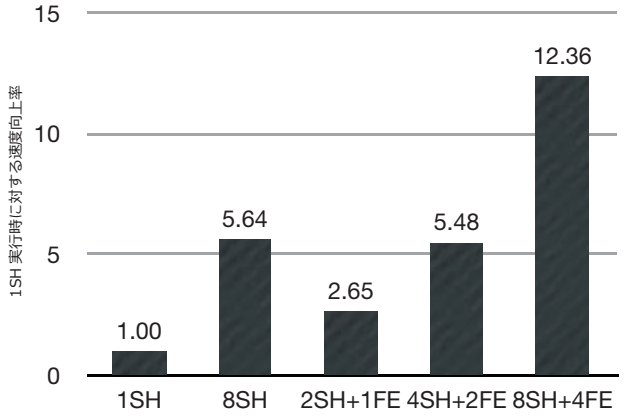

図 11 アクセラレータコンパイラによるバイナリを使用した場合の性能評価結果
Fig. 11 Experimental results(using accelerator compiler)

イラによるキャッシュ操作指示によりクラスタ間のコヒーレンシ制御を行っている[22]．本評価では汎用コアとして最大 8 基の SH-4A コア，アクセラレータコアとして使用する最大 4 基の FE-GA コアを計算資源として用いた．また，プログラム及び，データは命令キャッシュ及びデータキャッシュが有効なメモリ領域に配置した．

4.2 評価手法

3 章で述べたとおり，提案フレームワークでは，アクセラレータ用実行バイナリとして以下の 2 種類を用いることができる．

- アクセラレータコンパイラが生成した実行バイナリ
- 既存のアクセラレータ用ライブラリの使用

本節ではこれら 2 通りの方式により，提案フレームワークを評価した結果について報告する．

4.3 アクセラレータコンパイラによるバイナリを使用した場合

本評価では Optical Flow 計算プログラムを用いて性能評価を行った．Optical Flow 計算とは時間的に連続な画像を入力とし，移動した物体やカメラの速度場を求める計算のことである．本評価では OpenCV[23] ライブラリ内のブロックマッチング法を Parallelizable C 言語[24] で実装したプログラムを用いた．ブロックマッチング法では入力画像をブロックと呼ばれる単位に分割し，2 画像間で対応するブロックを探索して速度ベクトルを生成する．まず，入力の 2 画像を読み込み，その後前フレームの画像を Y 軸及び X 軸で走査し基準となるブロックを決定する．そして，注目フレームの画像をブロック単位で走査し，基準となるブロックと注目フレーム内のブロック間で差分計算を行い，差分が閾値以下になるブロックを見つけ，速度ベクトルを生成する．本アルゴリズムでは，各ブロックのマッチング処理を並列に行うことが可能であり，Y 軸を走査するループを分割し X 軸方向の処理を粗粒度タスクとしてヘテロジニアス並列化したコードを OSCAR コンパイラは生成する．また，FE-GA 用コンパイラによって差分計算を行う SAD 演算 (Sum of Absolute Difference) の部分が FE-GA で実行可能であると判定され，提案コンパイルフローに従い，実行バイナリが生成される．本評価での入力画像は 320x352(pixel) の画像であり．Optical Flow 計算時のブロックサイズは 16x16(pixel) である．なお，FE-GA へのデータ転送は CPU により行った．

図 11 に性能評価結果を示す． 図 11 において横軸はプロセッサ構成で，mSH+nFE は SH-4A コア m 基，FE-GA コア n 基を計算資源として使用したことを示す．縦軸は 1SH での実行時間に対する速度向上率である．8SH，8SH+4FE 構成時において，5.64 倍，12.36 倍と高い速度向上率を得ること，そして提案フレームワークが複数のプロセッサ構成において正しく動作することを確認した．

4.4 既存のライブラリを使用した場合

本評価では Optical Flow 計算および AAC エンコーダを用いて性能評価を行った．Optical Flow 計算プログラムは，株式会社日立製作所中央研究所および東北大学張山研究室により開発された[25] ものであり，それを Parallelizable C 言語で実装したプログラムを用いた．本プログラムでは上記の OpenCV 版アルゴリズムと同様に入力画像をブロックと呼ばれる単位に分割し，2 画像間で対応する点を探索して速度ベクトルを生成するが，探索をピクセル単位で行う点で OpenCV 版アルゴリズムと異なる．また，FE-GA 実行部分は前述同様 SAD 演算を行う部分であり，既存のライブラリを用いた．本評価での入力画像は 352x240(pixel) の画像であり．Optical Flow 計算時のブロックサイズは 16x16(pixel) である．なお，FE-GA へのデータ転送は CPU により行った．AAC エンコーダは株式会社ルネサステクノロジ及び株式会社日立製作所により提供された，AAC-LC エンコードプログラムを Parallelizable C 言語で実装したプログラムである．本アルゴリズムでは，入力された各フレームに対して，フィルタバンク，M/S ステレオ，量子化及びハフマン符号化を行う．各フレームのエンコード処理を並列に行うことが可能であり，1 つのフレームのエンコード処理を粗粒度タスクとしてヘテロジニアス並列化したコードを OSCAR コンパイラは生成する．フィルタバンク，M/S ステレオ及び量子化部分は FE-GA による高速化が可能であり，実行バイナリは株式会社日立製作所開発の既存のライブラリを用いた．本評価での入力音声は 19 秒の PCM ファイルであり，サンプリングレートは 44.1KHz，ビットレートは 128bps である．なお，FE-GA へのデータ転送は，転送データをプログラム開始時に分散共有メモリに配置し，DTU により行った．

図 12 に性能評価結果を示す． 図 12 において横軸はプロセッサ構成で，mSH+nFE は SH-4A コア m 基，FE-GA コア n 基を計算資源として使用したことを示す．縦軸は 1SH での実行時間に対する速度向上率である．8SH，8SH+4FE 構成時において，Optical Flow 計算で 5.4 倍，32 倍，AAC エンコーダで 6.3 倍，16 倍と高い速度向上率を得ること，そして提案フレームワークが複数のプロセッサ構成において正しく動作することを確認した．

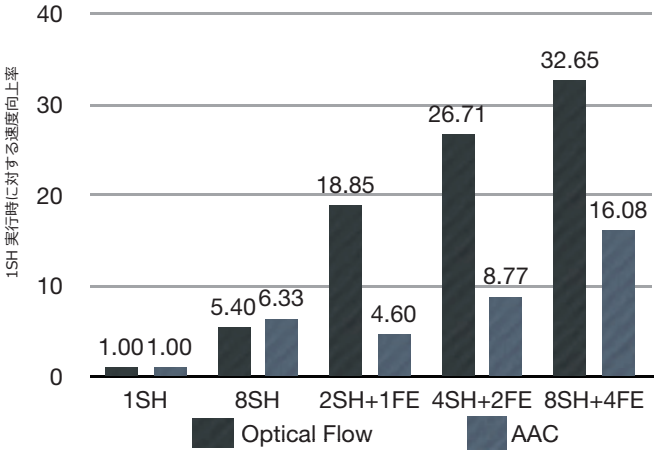

図 12 既存のライブラリを使用した場合の性能評価結果
Fig. 12 Experimental results(using hand-tuned library)

5. ま と め

本稿では複数 CPU 及びアクセラレータを持つヘテロジニアスマルチコアに対して，逐次プログラムを入力とし自動で最適な負荷分散を実現するため，ヘテロジニアスマルチコア向け OSCAR-API を提案し，それに基づいてフレームワークを構築した．本フレームワークでは自動並列化コンパイラ及びアクセラレータコンパイラを協調動作させることで，逐次 C プログラムを自動的にヘテロジニアス並列化し，高い性能を実現する．本手法を情報家電用ヘテロジニアスマルチコアプロセッサ RP-X をターゲットとして，AAC エンコーダ及び Optical Flow 計算の自動並列化性能を評価した．その結果，8 つの汎用 CPU コア及び 4 つのアクセラレータコアを使用した場合，逐次実行時と比較して Optical Flow 計算で約 12 倍（OSCAR コンパイラ+アクセラレータコンパイラ使用時），約 32 倍（OSCAR コンパイラ+既存ライブラリ使用時），AAC エンコーダで約 16 倍（OSCAR コンパイラ+既存ライブラリ使用時）の性能向上が得られ，ヘテロジニアスマルチコアを対象として，複数のプロセッサ構成に柔軟にコンパイル可能な汎用的なコンパイラフレームワークを実現可能であることがわかった．

謝辞 本研究の一部は，NEDO “ヘテロジニアスマルチプロセッサ” プロジェクト及び早稲田大学グローバルＣＯＥプログラム「アンビエントＳｏＣ教育研究の国際拠点」（文部科学省研究拠点形成費補助金）の支援により行われた．研究を遂行するにあたり，貴重なアドバイスを頂いたプロジェクト関係者の皆様，FE-GA 実行ライブラリをご提供いただいた東北大学張山研究室に感謝致します．

参 考 文 献

1) Pham, D., Asano, S., Bolliger, M., Day, M.N., Hofstee, H.P., Johns, C., Kahle, J., Kameyama, A., Keaty, J., Masubuchi, Y., Riley, M., Shippy, D., Stasiak, D., Suzuoki, M., Wang, M., Warnock, J., Weitzel, S., Wendel, D., Yamazaki, T. and Yazawa, K.: The design and implementation of a first-generation CELL processor, *2005 IEEE International Solid-State Circuits Conference, ISSCC* (2005).
2) Luebke, D., Harris, M., Govindaraju, N., Lefohn, A., Houston, M., Owens, J., Segal, M., Papakipos, M. and Buck, I.: GPGPU: General-purpose computation on graphics hardware, *2006 ACM/IEEE Conference on Supercomputing, SC'06* (2006).
3) Seiler, L., Carmean, D., Sprangle, E., Forsyth, T., Abrash, M., Dubey, P., Junkins, S., Lake, A., Sugerman, J., Cavin, R., Espasa, R., Grochowski, E., Juan, T. and Hanrahan, P.: Larrabee: A many-core x86 architecture for visual computing, *ACM Transactions on Graphics*, Vol.27, No.3 (2008).
4) Yoshida, Y., Kamei, T., Hayase, K., Shibahara, S., Nishii, O., Hattori, T., Hasegawa, A., Takada, M., Irie, N., Uchiyama, K., Odaka, T., Takada, K., Kimura, K. and Kasahara, H.: A 4320MIPS Four-Processor Core SMP/AMP with Individually Managed Clock Frequency for Low Power Consumption, *IEEE International Solid-State Circuits Conference, ISSCC* (2007).
5) Yuyama, Y., Ito, M., Kiyoshige, Y., Nitta, Y., Matsui, S., Nishii, O., Hasegawa, A., Ishikawa, M., Yamada, T., Miyakoshi, J., Terada, K., Nojiri, T., Satoh, M., Mizuno, H., Uchiyama, K., Wada, Y., Kimura, K., Kasahara, H. and Maejima, H.: A 45nm 37.3GOPS/W heterogeneous multi-core SoC, *IEEE International Solid-State Circuits Conference, ISSCC* (2010).
6) Masayasu, Y., Takeshi, S., Toshiaki, T., Yasuhiko, K. and Toshinori, I.: NaviEngine 1, System LSI for SMP-Based Car Navigation Systems, *NEC TECHNICAL JOURNAL*, Vol.2, No.4 (2007).
7) 木村, 藤井, 西道, 清原：デジタル家電統合プラットフォーム UniPhier におけるメディアプロセッサ，DA シンポジウム (2005).
8) Gelado, I., Kelm, J.H., Ryoo, S., Lumetta, S.S., Navarro, N. and Hwu, W. M.W.: CUBA: An architecture for efficient CPU/Co-processor data communication, *22nd ACM International Conference on Supercomputing, ICS'08*, pp.299–308 (2008).
9) Garland, M., Grand, S.L., Nickolls, J., Anderson, J., Hardwick, J., Morton, S., Phillips, E., Zhang, Y. and Volkov, V.: Parallel computing experiences with CUDA, *IEEE Micro*, Vol.28, No.4, pp.13–27 (2008).
10) AMD: ATI CTM Guide, http://ati.amd.com/companyinfo/researcher/documents/ATI_CTM_Guide.pdf.
11) khronos.org: OpenCL, http://www.khronos.org/opencl/.
12) Intel: Intel Threading Building Block, http://software.intel.com/en-us/intel-tbb/.
13) Luk, C., Hong, S. and Kim, H.: Qilin: Exploiting Parallelism on Heterogeneous Multiprocessors with Adaptive Mapping, Microarchitecture, *2009. MICRO-42. Proceedings. 42th Annual IEEE/ACM International Symposium on Microarchitecture* (2009).
14) Bellens, P., Perez, J.M., Badia, R.M. and Labarta, J.: CellSs: a programming model for the cell BE architecture, *In Proceedings of the 2006 ACM/IEEE Conference on Supercomputing(SC'06)* (2009).
15) Kasahara, H., Obata, M. and Ishizaka, K.: Automatic Coarse Grain Task Parallel Processing on SMP using OpenMP, *Proc of The 13th International Workship on Languages and Compilers for Parallel Computing(LCPC2000)* (2000).
16) Kimura, K., Mase, M., Mikami, H., Miyamoto, T. and Kasahara, J. S.H.: OSCAR API for Real-time Low-Power Multicores nad Its Performance on Multicores and SMP Servers, *Proc of The 22nd International Workship on Languages and Compilers for Parallel Computing(LCPC2009)* (2009).
17) 早大笠原・木村研究室：OSCAR-API v1.0, http://www.kasahara.cs.waseda.ac.jp/.
18) 和田, 林, 益浦, 白子, 中野, 鹿野, 木村, 笠原：ヘテロジニアスマルチコア上でのスタティックスケジューリングを用いた MP3 エンコーダの並列化，情報処理学会論文誌コンピューティングシステム (2007).
19) 本多, 岩田, 笠原：Fortran プログラム粗粒度タスク間の並列性検出法，信学論 (D-I), Vol.J73-D-I, No.12, pp.951–960 (1990).
20) 笠原, 合田, 吉田, 岡本, 本多：Fortran マクロデータフロー処理のマクロタスク生成手法，信学論, Vol.J75-D-I, No.8, pp.511–525 (1992).
21) 津野田, 高田, 秋田, 田中, 佐藤, 伊藤：ディジタルメディア向け再構成型プロセッサ FE-GA の概要，信学技報 RECONF2005-65 (2005).
22) 間瀬, 木村, 笠原：マルチコアにおける Parallelizable C プログラムの自動並列化，情処技報 ARC2009-174 (2009).
23) opencv.jp: OpenCV, http://opencv.jp/.
24) 間瀬, 馬場, 長山, 田野, 益浦, 宮本, 白子, 中野, 木村, 笠原：情報家電用マルチコア SMP 実行モードにおける制約付き C プログラムのマルチグレイン並列化，組み込みシステムシンポジウム (2007).
25) 張山, ハシタ, 奥村, 亀山：マルチメディア応用ヘテロジニアスマルチコアアーキテクチャの評価，信学技報 ICD2008-139 (2009).