情報処理学会論文誌　コンピューティングシステム　Vol.5　No.1　68–79 (Jan. 2012)

ヘテロジニアスマルチコア向け ソフトウェア開発フレームワークおよびAPI

林 明宏[1,a)]　和田 康孝[1]　渡辺 岳志[1]　関口 威[1]　間瀬 正啓[1]　白子 準[1]
木村 啓二[1]　笠原 博徳[1]

受付日 2011年5月11日, 採録日 2011年8月29日

概要：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をプログラマが記述する必要があるためプログラミングが困難である．そこで本論文では，逐次プログラムを入力とし自動並列化コンパイラを用いることで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，高い性能および低消費電力を実現可能なソフトウェア開発フレームワークを提案する．本手法はアクセラレータコンパイラやアクセラレータライブラリ等既存のアクセラレータ開発環境を有効に利用可能である．本フレームワーク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，アクセラレータライブラリを使用し，AAC エンコーダおよび Optical Flow 計算の自動並列化性能および消費電力を評価した．その結果，8 つの汎用 CPU コアおよび 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 計算で最大 32 倍，AAC エンコーダで最大 80%の電力を削減可能であることを確認し，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した．

キーワード：ヘテロジニアスマルチコア，自動並列化コンパイラ，マルチコア API，低消費電力化

Parallelizing Compiler Framework and API for Heterogeneous Multicores

Akihiro Hayashi[1,a)]　Yasutaka Wada[1]　Takeshi Watanabe[1]
Takeshi Sekiguchi[1]　Masayoshi Mase[1]　Jun Shirako[1]　Keiji Kimura[1]
Hironori Kasahara[1]

Received: May 11, 2011, Accepted: August 29, 2011

Abstract: There has been a growing interest in heterogeneous multicores because heterogeneous multicores achieve high performance keeping power consumption low. However, heterogeneous multicores force programmers very difficult programming. In order to overcome such a situation, this paper proposes a compilation framework which realizes high performance and low power. This paper also evaluates processing performance and the power reduction by the proposed framework on RP-X processor. The framework attains speedups up to 32x for an optical flow program with eight general purpose processor cores and four DRP (Dynamically Reconfigurable Processor) accelerator cores against sequential execution by a single processor core and 80% of power reduction for the real-time AAC encoding when we utilize an existing accelerator library.

Keywords: heterogeneous multicore, automatic parallelizing compiler, multicore API, low power

1. はじめに

1 チップ上に汎用プロセッサコアに加え特定処理を高い

[1] 早稲田大学
Waseda University, Shinjuku, Tokyo 169–8555, Japan
[a)] ahayashi@kasahara.cs.waseda.ac.jp

性能で実行可能なアクセラレータコアを搭載したヘテロジニアスマルチコアは組み込み分野からハイパフォーマンス分野まで広く注目を集めている．これは，アクセラレータの搭載により低消費電力で高い性能の実現が可能なためであり，従来までに CELL BE [1]，GPGPU [2]，RP1 [3]，RP-X [4] 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並列処理を行う粒度の決定，タスクの抽出，アクセラレータのプログラミング，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プログラマが負うことになり，並列プログラム開発のコストが問題となる．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ことが重要である．

この問題の緩和を目指し，アクセラレータプログラムの開発を容易化するものは数多く提案されている．たとえば GPGPU [2] では，NVIDIA は CUDA [5]，Khronos Group は OpenCL [6]，Portland Group は PGI Accelerator Compiler [7]，Caps Enterprise は HMPP [8] をそれぞれ提案している．これらはアクセラレータのプログラミングおよび，汎用コアとのインタフェースを API で抽象化しており，ユーザはアクセラレータ利用の恩恵を享受可能である．ヘテロジニアスマルチコアではこれらを有効に活用して，タスクスケジューリングによって汎用コアとの負荷分散をはかることが性能向上の鍵である．

負荷分散という観点からみれば，C.K Luk らは，アクセラレータプログラミングに CUDA [5] を利用し，学習アルゴリズムを利用して実行時に動的にタスクを割り当て，負荷の分散を実現するフレームワーク Qilin [9] を提案している．しかしながら，Qilin ではプログラマが手動でタスクの抽出を行う必要があり，根本的な問題解決には至っていない．自動負荷分散という観点から見れば，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 [10] 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はできていない．

以上をまとめると，ヘテロジニアスマルチコアを対象とした場合，アクセラレータ用プログラムの開発を容易化するものは数多く提案されているといえる．しかしながら，プログラム中で並列処理を行う粒度を決定し，タスクの抽出を行い，さらに，既存もしくは新規開発のアクセラレータ用ライブラリを活用し，汎用コアとアクセラレータの負荷分散およびコード生成まで自動で行うものは存在しない．この問題を解決するため，筆者らはヒント情報が含まれた逐次のプログラムから粒度の決定およびタスクを抽出しタスクスケジューリングを行い，かつ既存もしくは新規開発のアクセラレータライブラリ，およびアクセラレータコンパイラを組み合わせて，並列化コードを生成すること，つまり自動並列化を実現することで，プログラマが容易にヘテロジニアスマルチコアの性能を引き出すことが可能であると考えている．

筆者らはすでに，ヘテロジニアスマルチコア向けの並列性抽出，タスクスケジューリングおよびコード生成を行う OSCAR コンパイラ [11] を提案しており，シミュレータ上でその効果を確認している．また，ホモジニアスマルチコアに対しては様々な実チップ上でコンパイラによる自動並列化および低消費電力制御を実現可能とする OSCAR API [12] を提案している．OSCAR API [12] は OpenMP [13] の指示文をベースとし，電力制御やメモリ配置等の指示文を加えた組み込み向けマルチコア指示文の集合である．本論文でこれらを拡張し，既存のアクセラレータライブラリやアクセラレータコンパイラと OSCAR コンパイラを組み合わせて利用可能とするインタフェースを構築することで，ヘテロジニアスマルチコア用に汎用コアおよびアクセラレータへのタスク生成，スケジューリングおよびコード生成を行う自動並列化を可能とする．具体的には既存の OSCAR API をヘテロジニアスマルチコア用に拡張した OSCAR ヘテロジニアス API を含んだヘテロジニアスマルチコア向けのソフトウェア開発フレームワークを提案する．これにより

- 様々なプロセッサ構成に柔軟に対応可能なコンパイル手法
- アクセラレータ非依存でかつ既存のソフトウェア開発環境を活用可能なコンパイル体系

が実現可能である．加えて本論文では，提案体系のヘテロジニアスマルチコアチップ RP-X 上での性能評価について述べる．またこれらの実現により，筆者らが従来から提案しているコンパイラによる低消費電力化手法 [12] がヘテロジニアスマルチコアを対象として適用可能となったため，あわせて世界初のコンパイラによるヘテロジニアスマルチコアの低消費電力化手法の適用評価結果についても述べる．

2. 対象ヘテロジニアスマルチコアアーキテクチャおよびプログラム実行モデル

本章では，プロセッサ構成に依存しない柔軟なコンパイル体系および，既存のアクセラレータ環境を活用可能なコンパイル体系の実現に向けて，まずは本手法がコンパイル対象とするヘテロジニアスマルチコアアーキテクチャおよびその実行モデルを定義する．本論文ではアクセラレータの利用に必要な以下の処理を行う汎用プロセッサをコントローラと呼ぶ．

- アクセラレータへのデータ供給
- アクセラレータの起動
- アクセラレータからの演算結果の回収

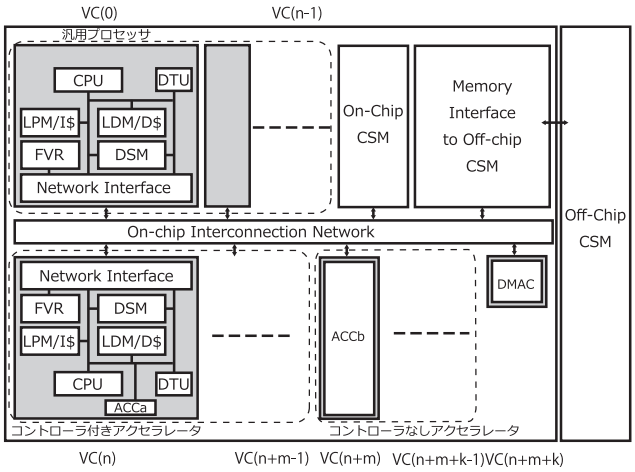


図 1　OSCAR API Applicable ヘテロジニアスマルチコアアーキテクチャ

Fig. 1　OSCAR API Applicable heterogeneous multicore architecture.

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

まず，提案手法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を**図 1** に示す．本アーキテクチャは複数個の汎用プロセッサおよびアクセラレータ，DMA コントローラ（DMAC）そしてチップ内/外の集中共有メモリ（CSM）から構成される．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，CPU と密結合されているもの（コントローラ付きアクセラレータと呼ぶ），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と呼ぶ）の両方をサポートしている．

また，汎用プロセッサおよび，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），電力制御レジスタ（FVR），命令キャッシュメモリ，データキャッシュメモリを持つことが可能である．LDM はコアプライベートのデータを格納するためローカルメモリ，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．このように，汎用コアとコントローラ付きアクセラレータの 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 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ラップ可能である．本アーキテクチャはデータ転送ユニット（DTU）やローカルメモリ（LDM），分散共有メモリ（DSM）を用いたローカリティの有効利用 [14]，そして電力制御レジスタ（FVR）を用いた電力制御をコンパイラが実現することにより，自動的に高性能かつ低消費電力を実現するコンパイラ協調型ヘテロジニアスマルチコアアーキテクチャである．

また，ヘテロジニアスマルチコアには汎用コアのほか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，フレームワークおよび 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，以下の例のようにコア種別ごとに連続した番号が割り振られる．本例は汎用コア n 基，コントローラ付きアクセラレータ m 基，コントローラなしアクセラレータ k 基および DMAC 等その他のコアを搭載したシステムへの VC 番号の付与例である．

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

2.2　汎用性を考慮したコンパイル体系構築のための役割分担の定義

本節では，前節のアーキテクチャに対して汎用的なコンパイル体系を実現することを目的に，既存のアクセラレータライブラリやアクセラレータコンパイラと OSCAR コンパイラの役割分担に関して議論を行う．

一般的に，ヘテロジニアスマルチコアにおいてアクセラレータを利用するには，プログラム中のどの部分がアクセラレータで実行可能であるかの判断を行うこと，アクセラレータ用のバイナリを生成すること，コントローラのコードを生成することが必要である．ただ，これらの処理は，対象とするアクセラレータにより様々であり，プログラミングモデルやアクセラレータの駆動方法，そして転送方式をすべて OSCAR コンパイラで対応するのは困難であり，汎用性・拡張性に欠けることになる．すでに述べたように，本提案では，汎用性を重要視しており，利用するアクセラレータに非依存なコンパイル体系の構築に注力している．そのため，汎用性を失わないように，OSCAR コンパイラがアクセラレータ実行可能箇所判定や，アクセラレータバイナリ生成，コントローラコードの生成を行うのではなく，アクセラレータコンパイラやアクセラレータライブラリ作成者がその任を負うこととする．

そのうえで，OSCAR コンパイラは主に，プログラムの並列性の抽出や，タスクスケジューリング，データの分割やメモリ管理に注力し，アクセラレータコンパイラはターゲットとなるアクセラレータに依存する部分の対応に注力するものとした．具体的にはアクセラレータコンパイラは以下の 3 つの要件を満たすものと定義する．

(1) 逐次プログラム中のアクセラレータ実行可能等の情報を OSCAR コンパイラ向けヒント指示文としてソースプログラム中に挿入．
(2) OSCAR コンパイラによるタスクスケジューリングの

結果，切り出されたアクセラレータで実行される関数をバイナリに変換．

(3) コントローラが行うアクセラレータの起動，データ供給および回収を行うソースコードの生成．

そして，OSCAR コンパイラは以下の 3 つの処理を主な役割とする．

(1) OSCAR コンパイラ向けヒント指示文を解釈し，タスクグラフの生成，タスクスケジューリング，メモリ管理を行う．

(2) スケジューリングの結果アクセラレータで実行されるタスクを入力ソースと同一の形で関数として切り出す．

(3) OSCAR API を生成する．

これらの作業の切り分けにより，後述するように，2 つのヒント指示文と 1 つの OSCAR API および 1 つの命名規則の追加で，アクセラレータ非依存の自動並列化コンパイル体系が構築可能である．

ただし，アクセラレータコンパイラの機能は様々で，日立による FE-GA [15] コンパイラのように上記の 3 点をすべて満たすものもあれば，Portland Group の PGI Accelerator Compiler [7] のように，アクセラレータ実行箇所の判定のみ不可能なもの，NVIDIA による CUDA コンパイラ [5] のように，アクセラレータバイナリの生成のみ可能なもの等がある．そのため，このような上記の機能をすべて満たしていないアクセラレータコンパイラを利用したい場合や，hand-tuning された既存のライブラリを使用する場合にも対応可能な手段も同時に提供している．この場合は，実行バイナリおよびコントローラのコードを，ハンドアセンブルや，アクセラレータコンパイルの使用等何からの形であらかじめアクセラレータライブラリとして用意しておき，ヒント指示文の挿入を手動で行うこととなる．本論文では，このような場合を「ライブラリアプローチ」と呼ぶ．

2.3 対象ヘテロジニアスマルチコアアーキテクチャ上でのプログラム実行モデルの定義

次に，2.1 節で述べたアーキテクチャ上でのプログラム実行モデルを規定する．スレッド実行モデルはワンタイムシングルレベルスレッド生成モデルであり，プログラム開始時に使用する汎用プロセッサおよびコントローラの個数分スレッドを生成し，そのままプログラム終了時までスレッドの生成および削除を行わない [12]．スレッドとの結びつきは静的に 1 対 1 の関係で決定される．

また，アクセラレータの制御方法は，コントローラの有無によって異なる．それぞれの実行モデルを**図 2**，**図 3** に示す．図 2，図 3 において，縦方向が時間軸を表し，コントローラがアクセラレータへのデータ供給を行った後，アクセラレータを起動し，その終了を待って，データの回収を行うことを示している．コントローラ付きアクセラレータの場合は，アクセラレータと同梱されている CPU がアクセラレータの制御を行うコントローラの役割を果たしている．この場合は，OSCAR コンパイラによって，ローカルメモリや分散共有メモリにロードされたデータをアクセラレータに転送する際，トラフィックがコントローラ付きアクセラレータの内部バスにとどまるため，高い性能向上が期待できる．コントローラなしアクセラレータの場合は，チップ内の汎用プロセッサのうちどれかをコントローラとして使用する．この場合は，OSCAR コンパイラによってコントローラとなる汎用プロセッサのローカルメモリや分散共有メモリにロードされたデータをコア間ネットワーク経由して転送することになり，場合によってはボトルネックになる可能性もある．

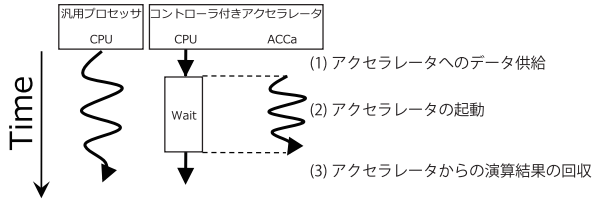


図 2　コントローラ付きアクセラレータ使用時の実行モデル
Fig. 2　Execution model on accelerator with controller.

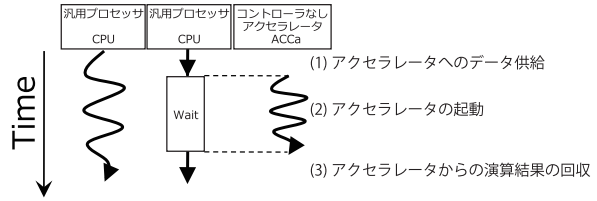


図 3　コントローラなしアクセラレータ使用時の実行モデル
Fig. 3　Execution model on accelerator without controller.

どちらの場合においてもコントローラの実行は，図中の “Wait” が示すようにアクセラレータの処理実行中はブロックされる．

3. ヘテロジニアスマルチコア向けソフトウェア開発フレームワーク

本章では，OSCAR コンパイラおよび提案ソフトウェアフレームワークの詳細について述べる．

3.1 本フレームワークの意義

2.1 節で述べたヘテロジニアスなアーキテクチャに対して実チップをターゲットとして汎用的なコンパイル手法を構築するために，2.2 節において，OSCAR コンパイラとアクセラレータコンパイラの役割分担を定義した．すでに述べたように，本フレームワークでは，特に「OSCAR コンパイラ」と「(1) アクセラレータ実行可能個所指定，および (2) アクセラレータのバイナリ生成方法およびコントローラコードの生成方法」間のインタフェースを定義することにその意義がある．以下では，その役割分担をコンパイルフローとして定義し，具体的なコードを交えて詳細を

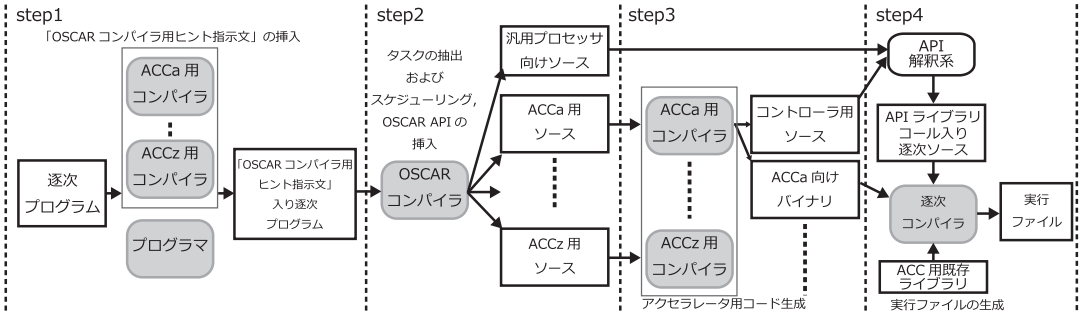


図 4　提案フレームワークのコンパイルフロー
Fig. 4　Compilation flow of the proposed framework.

説明する．

3.2　コンパイルフロー

図 4 に提案するフレームワークによるコンパイルフローを示す．本フローの入力は Parallelizable C [16] もしくは Fortran 77 で書かれた逐次プログラムで出力はターゲットアーキテクチャの実行バイナリである．Parallelizable C は並列性の解析を容易にする C 言語のコーディングルールであり，ポインタの使用法や再帰呼び出し等に関しての制限を加えている．本コンパイルフローの処理の流れは以下のようになる．

step 1：　アクセラレータ（ACC）コンパイラは逐次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等の情報を「OSCAR コンパイラ向けヒント指示文」としてソースプログラム中に挿入する．ライブラリアプローチの場合はプログラマが手動でヒント指示文の挿入を行う．

step 2：　OSCAR コンパイラは OSCAR コンパイラ向けヒント指示文入りソースプログラムを入力とし，粗粒度タスクのスケジューリング [11] や低消費電力制御 [12] を行い，その結果を「OSCAR API」入り C もしくは Fortran プログラムとして出力する．この OSCAR API は，従来の OSCAR API 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．このとき，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実行部分を関数として切り出し，これらを汎用プロセッサ用コードとは別ファイルに出力する．ライブラリアプローチの場合は，OSCAR コンパイラは切り出した関数の中に後述の命名規則に従ってライブラリコールを埋め込む．

step 3：　アクセラレータコンパイラは OSCAR コンパイラが切り出した関数をアクセラレータ用バイナリに変換すると同時にコントローラが行うアクセラレータの駆動，データ供給および回収を行うソースコードも生成する．ライブラリアプローチの場合は 2.2 節で述べたように，ライブラリ中にアクセラレータの駆動およびデータの供給・収集を行うコードが含まれていることを前提としているため，本ステップはなにもせずに通過する．

step 4：　ターゲットとなるヘテロジニアスマルチコア向けに用意された API 解釈系が OSCAR API 指示文を pthread 等のランタイムライブラリコールに変換する．最後に既存の逐次コンパイラを用いて実行ファイルが生成される．ライブラリアプローチの場合はこのフェーズでアクセラレータライブラリのリンクを行う．

```
#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 5　OSCAR コンパイラ向けヒント指示文
Fig. 5　Hint directive for OSCAR compiler.

3.3　OSCAR コンパイラ向けヒント指示文

本節ではプログラム中のアクセラレータ実行可能部分の情報を OSCAR コンパイラに与える OSCAR コンパイラ向けヒント指示文について説明する．前節 step 1 で挿入される OSCAR コンパイラ向けヒント指示文を**図 5** に示す．また，図中の “[ ]” は省略可能であることを示す．

accelerator_task はアクセラレータによって実行可能なブロックを OSCAR コンパイラに通知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 として与えることで，OSCAR コンパイラはタスクスケジューリング時のタスクコストとして使用する．コントローラとアクセラレータ間のデータ転送方法を trans_mode で指定することで，OSCAR コンパイラはタスクスケジューリング時に DTU や DMA 等が busy 状態かどうかを考慮可能となる．たとえば，ある accelerator_task の trans_mode に DTU があらかじめ指定された場合，OSCAR コンパイラのデータ転送スケジューリングでは，当該タス

```
int main() {
  int i, x[N], var1 = 0;
  /* loop1 */
  for (i = 0; i < N; i++) { x[i] = i; }
  /* loop2 */
#pragma oscar_hint accelerator_task (ACCa) \
            cycle(1000,((OSCAR_DMAC()))) workmem(OSCAR_LDM(), 10)
  for (i = 0; i < N; i++) { x[i]++; }
  /* function3 */
#pragma oscar_hint accelerator_task (ACCb) \
            cycle(100, ((OSCAR_DTU()))) 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);  //hand-tuned library call
}
```

図 6　OSCAR コンパイラ向けヒント指示文例
Fig. 6　Example of source code with hint directives.

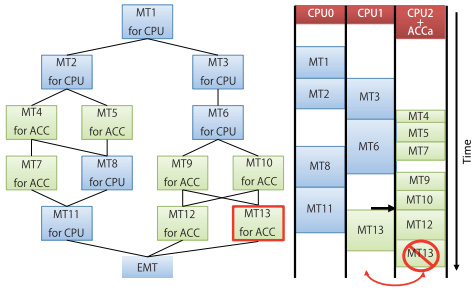


図 7　タスクスケジューリング例
Fig. 7　Example of task scheduling.

ク実行中はコントローラと他の汎用コア間のデータ転送には DTU を使用しないというスケジューリングが可能となる．また，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．そして，コントローラがアクセラレータを制御する際に作業用に使用するメモリ種別と容量を mem_type と mem_size で指定可能である．これは，アクセラレータを使用する際にソースコードに陽に示されずに使用されるデータの量を OSCAR コンパイラに通知することで，OSCAR コンパイラはローカルメモリ管理を行う際に，どのメモリをどれだけ使用するかを把握し，メモリ管理が可能である．

最後に，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で，ソースが提供されていない等の理由でデータ依存解析が解析不能な場合に，OSCAR コンパイラがタスクグラフを作成する一助となる．これらの引数は OSCAR コンパイラのヒント情報であり，2.2 節で述べたとおり，コントローラとアクセラレータ間のデータ転送コードはアクセラレータコンパイラが生成することになる．また，すでに述べたように，本論文のフォーカスでは，OSCAR コンパイラが直接アクセラレータへのデータ転送コードを生成することはないが，将来的にこの in/out 指定を，アクセラレータがどのようなデータを入出力するのかというヒント指示文として用いることで，アクセラレータへのデータ転送も OSCAR コンパイラが管理することも可能である．

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等といった用途を想定している．たとえば，Portland Group の PGI コンパイラの acc_region ディレクティブのような指示文を直接指定する場合は，本指示文を用いる．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．

サンプルコードを図 6 に示す．サンプルコードは 2 つの “accelerator_task” および 1 つの “oscar_comment” のヒント指示文を含んでいる．この例では/* loop2 */以下のループがアクセラレータ ACCa で，/* function3 */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．/* loop2 */に付くアクセラレータタスク指示文は「ACCa で 1,000 クロックサイクル（データ転送コストを含む）で実行可能であり，データ転送には DMAC を利用する．そしてコントローラがアクセラレータの制御用に 10 バイトの作業用データを LDM に配置する」ということを示す．作業用のデータとは，コントローラがアクセラレータを制御する際に暗黙的に使用されるデータのことを指し．たとえば FE-GA [15] の場合，FE-GA へのデータ供給および回収時に特定のメモリに転送命令を配置する必要があるため，このような指定を行っている．同様に/* funtion3 */に付く指示文は，「ACCb で 100 クロックサイクル（データ転送コストを含む）で実行可能であり，データ転送には DTU を利用する．このタスクは変数 var1 と配列 x の 2 番目の要素から 11 番目の要素を入力として，配列 x の 2 番目の要素から 11 番目の要素が出力する」ということを示す．

関数 call_FFT() 内の 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．また，以下 call_FFT() 内の FFT() 関数はライブラリアプローチで使用されるアクセラレータライブラリであるとする．

3.4　OSCAR ヘテロジニアス並列化コンパイラ

本節では OSCAR コンパイラによるヘテロジニアス並列化について述べる．まず 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出した結果を階層的なマクロタスクグラフ（MTG）として表現する [17], [18], [19]．次に，OSCAR コンパイラはチップ上の計算資源の特性を考慮しながら，MTG 中の各 MT を静的にスケジューリングする [11]．図 7 は MTG を 2 基の汎用プロセッサおよび 1 基のコントローラ付きアクセラレータにスケジューリングする例を示す．スケジューラはすべてのタスクを割り当て終えるまで，タスクの依存関係を考慮しながらレディタスクを選び，データ転送のオーバ

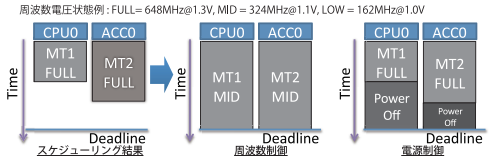


図 8　コンパイラによる電力制御例

Fig. 8　Example of power reduction by compiler.

ヘッドを考慮しながら演算資源に割り当てていく．図 7 内の MT13 のように，アクセラレータに多くのタスクが割り当てられている状態の場合は処理終了時間を最小化するために汎用プロセッサコアにアクセラレータ実行可能タスクを割り当てることもありうる．

そして，OSCAR コンパイラは MT 実行時の周波数および電圧の制御および電源遮断制御を行うことで消費電力の最小化を行う [20]．この際，コンパイラはタスクのスケジューリング結果をもとに，制御オーバヘッドやリアルタイム処理時のデッドラインを考慮しながら適切に電力制御命令の生成を行う（**図 8**）．デッドラインの指定はコンパイラへのディレクティブを使用する．図 8 において FULL は最大の動作周波数で動作していることを，MID はその半分の動作周波数で動作していることを示す．

最後に OSCAR コンパイラは OSCAR API が挿入された C もしくは Fortran コードを生成する．3.2 節で述べたとおり，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実行部分を関数として切り出し，これらを汎用プロセッサ用コードとは別に出力する．アクセラレータコンパイラはこの関数をアクセラレータ用バイナリに変換すると同時にコントローラが行うアクセラレータの駆動，データ供給および回収を行うソースコードも生成する．

3.5　OSCAR API のヘテロジニアスマルチコア向け拡張

OSCAR API のヘテロジニアスマルチコア向けの拡張の目的は，「OSCAR コンパイラ」と「アクセラレータのバイナリ生成およびコントローラコードの生成」間のインタフェースの定義である．

インタフェースの定義にあたり，考慮すべき点は，3 点ある．まず 1 点目はコントローラコードを生成するアクセラレータコンパイラに対して，当該アクセラレータのコントロールを行うコア番号の通知をどのように行うかという点，2 点目は OSCAR コンパイラがアクセラレータ用に切り出したファイル内の関数間で呼び出し関係がある場合，どの関数がアクセラレータ実行のエントリポイントとなるのかをどのように指定するのかという点，3 点目はコントローラがアクセラレータの終了をどのように判定するかという点である．

1 点目のアクセラレータコンパイラに対するコントローラのコア番号の通知に関しては，自コアと他コアの両方からアクセス可能な分散共有メモリからアクセラレータにデータ供給を行う際に，コア内ローカルバスを経由した自コア内アクセス，コア間ネットワークからのアクセスをアクセラレータコンパイラが判定し，最適なアドレス生成をするときに必要となる．

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

図 9　ヘテロジニアスマルチコア用 OSCAR API

Fig. 9　OSCAR API for heterogeneous multicore.

2 点目のアクセラレータ実行のエントリポイント指定に関しては，アクセラレータコンパイラがコントローラ用にアクセラレータの起動，データ転送および回収を行うコードを作成するために必要である．

3 点目の終了判定に関しては，2.3 節の実行モデルの説明で述べたように，コントローラの実行がアクセラレータ処理実行中にはブロックされる．アクセラレータの終了判定は，当該関数の呼び出しが終わり，制御がコントローラに戻ることで判定可能である．

1 点目のアクセラレータのコントロールを行うコア番号の通知，2 点目のアクセラレータ実行のエントリポイントになる関数の通知に関しては，アクセラレータも含めて並列処理の粒度を決定し，タスクスケジューリングを行う OSCAR コンパイラがその情報を保持しており，アクセラレータコンパイラで判定することは不可能のため，以下のヘテロジニアスマルチコア向け OSCAR API を 1 つ定義した（**図 9**）．なお OSCAR API では pragma の後は “oscar” となる．

本 API は callee で指定した関数あるいはサブルーチンを VCno で指定された汎用プロセッサから呼び出すように指定している．アクセラレータコンパイラは，callee のバイナリを生成し，VCno で指定されたコントローラ用にアクセラレータの起動，データ供給および回収を行うソースコードを生成する．ライブラリアプローチの場合は，OSCAR コンパイラはスケジューリングの結果に応じて関数名に VCx がコントローラプロセッサで VCy のアクセラレータを制御するという意味の oscarlib_CTRLx_ACCELy_という prefix をつける．コントローラとアクセラレータの組に応じたライブラリを上記の命名規則に従いあらかじめ用意しておき，リンク時に実体をリンクして実行バイナリを生成することになる．

図 6 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），および 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 の駆動等の制御を行う．**図 10** に OSCAR コンパイラが出力する汎用コア用のコード，VC2，VC3 用のコードをそれぞれ示す．図 10 では，main() 関数中で汎用

```
int main() {
#pragma omp parallel sections
  {
#pragma omp section
    { MAIN_CPU0(); }
#pragma omp section
    { MAIN_CPU1(); }
#pragma omp section
    { MAIN_CPU2(); }
  }
  return 0;
}
```

汎用 CPU 用ソース

```
int MAIN_CPU1() {
  //ACCb コントローラのスレッド
  oscartask_CTRL1_call_FFT(var1, &x);
  ...
}
int MAIN_CPU2() {
  //ACCa コントローラのスレッド
  oscartask_CTRL2_call_loop2(&x);
  ...
}
```

```
#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]++; }
}
```

ACCa 向けソース

```
#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);
}
```

ACCb 向けソース

図 10　ヘテロジニアスマルチコア API 出力例
Fig. 10　Example of parallelized source code with OSCAR API.

コア用のスレッドを MAIN_CPU0(), MAIN_CPU1(), および 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 10 では，accelerator_task_entry により CPU から呼び出される oscartask_CTRL2_loop2() が指定されており，アクセラレータコンパイラを使用する際の例となる．アクセラレータ用コンパイラは，この accelerator_task_entry によって指定された情報をもとに，コントローラコードおよびバイナリを生成する．ライブラリアプローチの例として，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，すでに述べた命名規則によって OSCAR コンパイラが命名し，あらかじめユーザが準備する，アクセラレータ用のライブラリ関数である．また，図 6 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

本章ではメディアアプリケーションを提案フレームワークによりコンパイルし，NEDO 情報家電用ヘテロジニアスマルチコアプロジェクトで開発した RP-X [4] 上で評価した結果について述べる．本評価の目的は提案するフレームワークにおける OSCAR コンパイラおよび OSCAR ヘテロジニアス API とアクセラレータコンパイラあるいはアクセラレータ用ライブラリの連携による，性能向上および低消費電力の有効性の確認である．

4.1 ヘテロジニアスマルチコア RP-X

RP-X は日立製作所，ルネサステクノロジ，東京工業大学，早稲田大学で共同開発された 45 nm Low Power テクノロジ，15 コアのヘテロジニアスマルチコアで，汎用コアとして 648 MHz で動作する SH-4A コアを 8 基，アクセラレータコアとして 324 MHz で動作する FE-GA [15] を 4 基，その他種々のハードウェア IP を搭載している（**図 11**）．

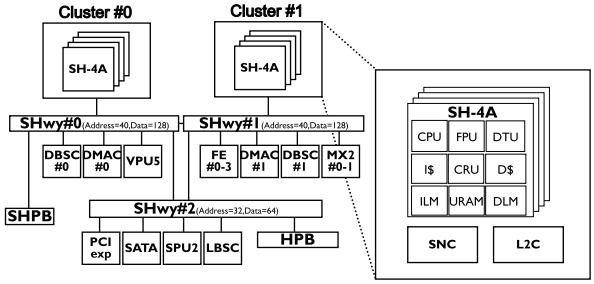


図 11　情報家電用ヘテロジニアスマルチコア RP-X
Fig. 11　RP-X heterogeneous multicore for consumer electronics.

各汎用プロセッサ内メモリは命令キャッシュ（32 KB），データキャッシュ（32 KB），ローカルメモリ（ILM，DLM: 16 KB），分散共有メモリ（URAM: 64 KB），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．

低消費電力制御向けの機能としては汎用プロセッサは DVFS およびクロックゲーティングが可能であり，アクセラレータは DVFS のみ可能である．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．アクセラレータのバイナリ生成は FE-GA コンパイラによって，あるいは既存のライブラリを利用する．電力評価においては，プロセッサボード上のテストピンを利用して CPU コアにかかる電圧，電流を KEYENCE 社製プローブによって測定し，電力制御を行った際の電力の評価を行う．

4.2 RP-X 上での処理性能

本評価ではアクセラレータコンパイラを用いたバイナリ生成の例として OpenCV [21] に含まれるブロックマッチング法による Optical Flow 計算プログラム，ライブラリアプローチの例として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 [22] Optical Flow 計算プログラム，そして，株式会社ルネサステクノロジお

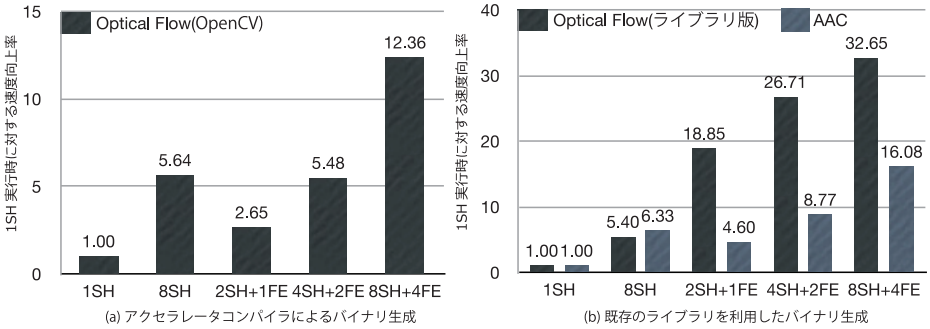


図 12　RP-X 上でのアプリケーションの性能評価結果
Fig. 12　Perfomance result on RP-X.

よび株式会社日立製作所により開発された，AAC-LC エンコードプログラムをそれぞれ逐次の Parallelizable C 言語で実装したプログラムを用いる．Optical Flow 計算とは時間的に連続な画像を入力とし，それをブロックと呼ばれる小領域に分割し，ブロック間で差分計算を行うことで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．入力画像サイズは OpenCV 版で 320 × 352(pixel)，ライブラリ版で 352 × 240(pixel)，ブロックサイズは 16 × 16(pixel) であり，どちらも差分計算である SAD（Sum of Absolute Difference）を計算する部分が FE-GA で実行可能である．OSCAR コンパイラはブロックごとに差分計算を実行する部分を粗粒度タスクとし，それを SH-4A および SH-4A と FE-GA の組に対して割り当てる．単一の SH-4A コアに対する速度向上率はそれぞれ，OpenCV 版で 2.3 倍，ライブラリ版で 25 倍である．性能評価にあたっては，OpenCV 版で 2 枚の画像データを処理する時間，ライブラリ版では 450 枚の画像データを処理する時間を性能評価の指標とした．

AAC エンコーダは，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，量子化およびハフマン符号化を行う．フィルタバンクと M/S ステレオおよび量子化部分は FE-GA で実行可能である．OSCAR コンパイラは 1 フレームのエンコード処理を行う部分を粗粒度タスクとし，それを SH-4A および SH-4A と FE-GA の組に対して割り当てる．単一の SH-4A コアに対する速度向上率はフィルタバンクと M/S ステレオで 24 倍，量子化部分で 7.7 倍である．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 44.1 KHz，ビットレートは 128 bps である．性能評価にあたっては，19 秒の PCM データをエンコーディングする時間を性能評価の指標とした．

また，アクセラレータコンパイラを利用する OpenCV 版では逐次プログラムに FE-GA コンパイラによる OSCAR コンパイラ向けヒント指示文が挿入され，ライブラリを利用する Optical Flow 計算および AAC ではユーザによるヒント指示文が挿入される．どちらの場合もその後 OSCAR コンパイラによる自動並列化が行われた後，定義されたコンパイルフローに従って実行バイナリが生成される．

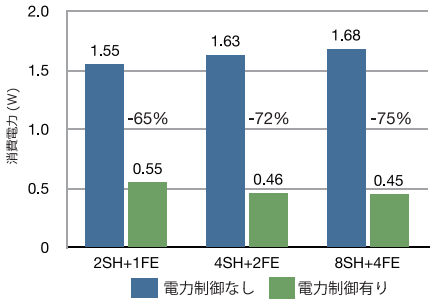


図 13　OSCAR コンパイラによる電力削減効果
Optical Flow 計算（ライブラリ版）
Fig. 13　Power reduction for Optical Flow.

図 12 に性能評価結果を示す．図中左部の (a) はアクセラレータコンパイラ用いた場合の Optical Flow 計算（OpenCV 版）の性能評価結果，右部の (b) は既存のアクセラレータライブラリを用いた場合の Optical Flow 計算（ライブラリ版）および AAC エンコーダの性能評価結果である．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．アクセラレータコンパイラを使用した場合は逐次実行に比べて最大 12 倍，既存のライブラリを使用した場合は Optical Flow 計算（ライブラリ版）で最大 32 倍，そして AAC エンコーダで最大 16 倍と高い性能向上を得，複数のプロセッサ構成において動作することを確認した．

4.3　RP-X 上での消費電力

本評価では前節の評価アプリケーションの既存のライブラリを用いた Optical Flow 計算（ライブラリ版）および AAC エンコーダにリアルタイム処理のデッドライン制御モードでの自動電力削減を行った場合の消費電力の評価を行った．デッドラインは Optical Flow 計算においては 1 つの画像の処理に 33 [ms]，つまり 30 [fps] を設定し，AAC エンコーダにおいては 1 秒間の音声データをエンコードす

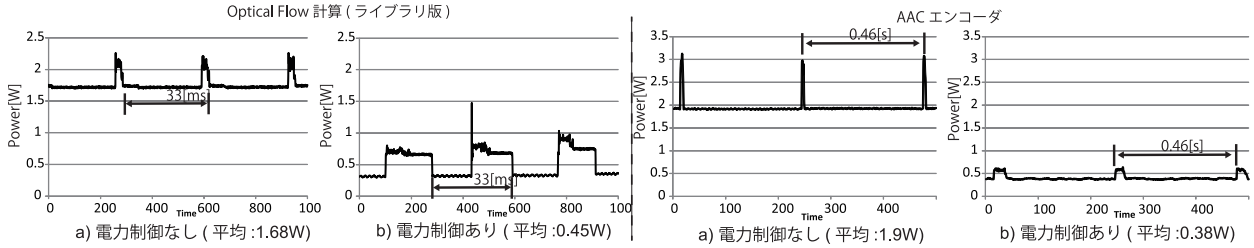


図 14　8SH+4FE 構成時の Optical Flow（ライブラリ版）と AAC の電力波形
Fig. 14　Power wave of Optical Flow and AAC.

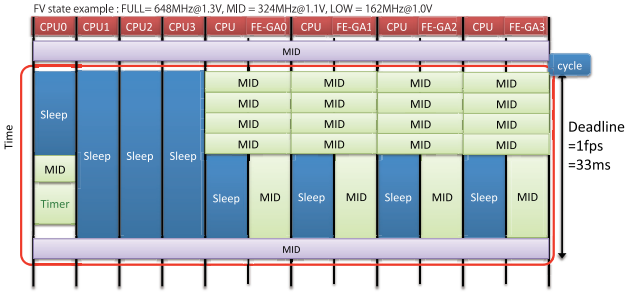


図 15　8SH+4FE 構成時の電力制御イメージ
Fig. 15　Example of power control for 8SH+4FE.

るのに 1 秒でエンコード可能なように設定した．

図 13 に Optical Flow 計算（ライブラリ版）の電力評価結果を示す．OSCAR コンパイラにより各プロセッサ構成で 65%～75%の電力削減を実現可能であり，最大 8SH+4FE 構成時に 1.68 [W] から 0.45 [W] と 75%の電力削減を確認した．プロセッサコアが増加することでデッドラインまでの余裕が増大し，各プロセッサをより低電力モードで実行可能なため，結果的に消費電力をより低減可能であることも分かる．また，**図 14** に 8SH+4FE 構成時の電力波形を示す．図 14 において横軸は経過時間であり，縦軸は消費電力 (W) である．図中矢印で示されたものは 1 枚の画像の処理に許されるデッドラインであり，左の 2 つのグラフが Optical Flow 計算に該当する．3.4 節で述べたコンパイラによる周波数および電圧の制御，クロックゲーティング等の API 生成により，消費電力が 1.68 [W] から 0.45 [W] に削減されていることが分かる．**図 15** は 8SH+4FE 構成時における電力制御のイメージを示している．図中の四角は粗粒度タスク（MT）を示しており，MT を実行する際の電力状態として，FULL は 648 MHz@1.3 V，MID は 324 MHz@1.1 V，LOW は 162 MHz@1.0 V である．図 15 より，各 FE-GA に対してそれぞれ 4 つのタスク（SAD 演算）が割り当てられており，各タスクは MID 状態で実行されていることが分かる．そして，リアルタイム処理用に時間の管理をしている CPU0 以外の CPU コアはデッドラインに到達するまでクロックゲーティング状態（図中の Sleep に該当）となっている．また，同様に AAC エンコーダにおいても図 14 の右の 2 つのグラフにより，1.9 [W] から 0.38 [W] と 80%の自動電力削減が得られたことが分かる．

5.　まとめ

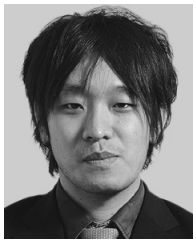
本論文では複数汎用 CPU コアおよびアクセラレータを持つヘテロジニアスマルチコアに対して，逐次プログラムを入力とし自動で最適な並列処理を実現するため，ヘテロジニアスマルチコア向け OSCAR API の提案および，それに基づくコンパイラフレームワークを構築した．本フレームワークでは逐次のプログラムを自動並列化し，かつ既存のアクセラレータライブラリ，およびアクセラレータコンパイラを有効に活用することでプログラマが容易にヘテロジニアスマルチコアの性能を引き出すことが可能である．本手法を情報家電用ヘテロジニアスマルチコアプロセッサ RP-X をターゲットとして，AAC エンコーダおよび Optical Flow 計算の自動並列化性能および消費電力を評価した．その結果，8 つの汎用 CPU コアおよび 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 計算で最大 32 倍，AAC エンコーダで 80%の電力を削減可能であることを確認し，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分かった．今後の課題としては様々なヘテロジニアスマルチコアに対して提案手法を適用し，有用性を評価することを考えている．

謝辞　本研究の一部は，NEDO “情報家電用ヘテロジニアスマルチプロセッサ” プロジェクトおよ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．OSCAR ヘテロジニアス API は早大と日立・ルネサス・富士通・東芝・パナソニック・NEC からなる早大 API 委員会で策定された．研究を遂行するにあたり，貴重なアドバイスをいただ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いたします．

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 de-

sign and implementation of a first-generation CELL processor, *IEEE International Solid-State Circuits Conference, ISSCC*, Vol.1, pp.184–592 (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, *Proc. 2006 ACM/IEEE Conference on Supercomputing* (*SC'06*), p.208, (2006).
[3] 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*, pp.100–590 (2007).
[4] 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*, pp.100–101 (2010).
[5] 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).
[6] khronos.org: OpenCL, available from ⟨http://www.khronos.org/opencl/⟩.
[7] Wolfe, M.: Implementing the PGI Accelerator model, *Proc. 3rd Workshop on General-Purpose Computation on Graphics Processing Units* (*GPGPU '10*) (2010).
[8] Dolbeau, R., Bihan, S. and Bodin, F.: HMPP(TM): A Hybrid Multi-core Parallel Programming Environment, *Proc. 1st Workshop on General Purpose Processing on Graphics Processing Units* (*GPGPU '07*) (2007).
[9] Luk, C., Hong, S. and Kim, H.: Qilin: Exploiting Parallelism on Heterogeneous Multiprocessors with Adaptive Mapping, Microarchitecture, *2009. MICRO-42. Proc. 42th Annual IEEE/ACM International Symposium on Microarchitecture*, pp.45–55 (2009).
[10] Bellens, P., Perez, J.M., Badia, R.M. and Labarta, J.: CellSs: A programming model for the cell BE architecture, *Proc. 2006 ACM/IEEE Conference on Supercomputing* (*SC'06*), p.5 (2009).
[11] 和田康孝，林　明宏，益浦　健，白子　準，中野啓史，鹿野裕明，木村啓二，笠原博徳：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，Vol.1, No.1, pp.105–119 (2008).
[12] Kimura, K., Mase, M., Mikami, H., Miyamoto, T. and Kasahara, J.S.H.: OSCAR API for Real-time Low-Power Multicores and Its Performance on Multicores and SMP Servers, *Proc. 22nd International Workship on Languages and Compilers for Parallel Computing* (*LCPC2009*) (2009).
[13] OpenMP.org: OpenMP, available from ⟨http://www.openmp.org/⟩.
[14] 中野啓史，桃園　拓，間瀬正啓，木村啓二，笠原博徳：マルチコアプロセッサ上での粗粒度並列処理のためのローカルメモリ管理手法，情報処理学会論文誌　コンピューティングシステム，Vol.2, No.2, pp.63–74 (2009).
[15] 津野田賢伸，高田雅士，秋田庸平，田中博志，佐藤真琴，伊藤雅樹：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).
[16] Mase, M., Onozaki, Y., Kimura, K. and Kasahara, H.: Parallelizable C and Its Performance on Low Power High Performance Multicore Processors, *Proc. 15th Workshop on Compilers for Parallel Computing* (2010).
[17] Kasahara, H., Obata, M. and Ishizaka, K.: Automatic Coarse Grain Task Parallel Processing on SMP using OpenMP, *Proc. 13th International Workshop on Languages and Compilers for Parallel Computing* (*LCPC2000*) (2000).
[18] 本多弘樹，岩田雅彦，笠原博徳：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).
[19] 笠原博徳，合田憲人，吉田明正，岡本雅巳，本多弘樹：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).
[20] Shirako, J., Oshiyama, N., Wada, Y., Shikano, H., Kimura, K. and Kasahara, H.: Compiler control power saving scheme for multi core processors, Lecture Notes in Computer Science 4339, pp.362–376 (2007).
[21] opencv.org: OpenCV, available from ⟨http://opencv.org/⟩.
[22] 張山昌論，ウィシディスーリヤ ハシタ ムトゥマラ，奥村大輔，亀山充隆：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).



林 明宏　（正会員）

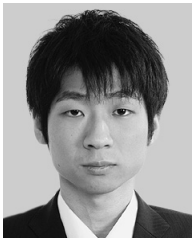
昭和59年生．平成19年早稲田大学理工学部コンピュータ・ネットワーク工学科卒業．平成20年同大学大学院理工学研究科情報理工学専攻修士課程修了．平成20年同大学院基幹理工学研究科情報理工学専攻博士後期課程進学．平成22年早稲田大学基幹理工学部情報理工学科助手，現在に至る．

和田 康孝　（正会員）

昭和54年生．平成14年早稲田大学理工学部電気電子情報工学科卒業．平成16年同大学大学院理工学研究科電気工学専攻修士課程修了．平成19年同大学院理工学研究科情報・ネットワーク専攻博士課程退学（満期）．平成21年博士（工学）早稲田大学．平成18年早稲田大学理工学部助手．平成19年同大学理工学術院基幹理工学部助手．平成21年同大学IT研究機構アドバンストマルチコアプロセッサ研究所次席研究員．平成22年同大学理工学術院基幹理工学研究科情報理工学専攻助教・エジプト日本科学技術大学情報・コンピュータ工学専攻特任准教授，現在に至る．

渡辺 岳志

昭和 61 年生．平成 20 年早稲田大学理工学部コンピュータ・ネットワーク工学科卒業．平成 22 年同大学大学院基幹理工学研究科情報理工学専攻修士課程修了．富士通株式会社入社，現在に至る．



関口 威

昭和 61 年生．平成 21 年早稲田大学理工学部コンピュータ・ネットワーク工学科卒業．平成 23 年同大学大学院基幹理工学研究科情報理工学専攻修士課程修了．パナソニック株式会社入社，現在に至る．

間瀬 正啓　（正会員）

昭和 58 年生．平成 17 年早稲田大学理工学部電気電子情報工学科卒業．平成 19 年同大学大学院理工学研究科情報・ネットワーク専攻修士課程修了．平成 19 年同大学院基幹理工学研究科情報理工学専攻博士後期課程進学．平成 20 年早稲田大学基幹理工学部情報理工学科助手．平成 23 年早稲田大学基幹理工学研究科情報理工学専攻博士（工学）学位取得．平成 23 年株式会社日立製作所入社，現在に至る．

白子 準　（正会員）

昭和 54 年生．平成 14 年早稲田大学理工学部電気電子情報工学科卒業．平成 19 年同大学大学院理工学研究科情報ネットワーク専攻博士課程修了．博士（工学）．平成 17 年同大学同学部助手．平成 19 年学術振興会特別研究員（PD）．平成 20 年アメリカ合衆国 Rice 大学ポスドク研究員．平成 22 年同大学研究員，現在に至る．

木村 啓二　（正会員）

昭和 47 年生．平成 8 年早稲田大学理工学部電機工学科卒業．平成 13 年同大学大学院理工学研究科電気工学専攻博士課程修了．平成 11 年早稲田大学理工学部助手．平成 16 年同大学理工学部コンピュータ・ネットワーク工学科専任講師．平成 17 年同助教授．平成 19 年同大学情報理工学科准教授，現在に至る．マルチコアプロセッサのアーキテクチャとソフトウェアに関する研究に従事．

笠原 博徳　（正会員）

昭和 32 年生．昭和 55 年早稲田大学理工学部電気工学科卒業，昭和 60 年同大学大学院博士課程修了，工学博士，昭和 61 年早稲田大学理工学部専任講師，平成 9 年教授，現在，情報理工学科教授，アドバンストマルチコアプロセッサ研究所長．昭和 60 年カリフォルニア大学バークレー校，平成元年イリノイ大学 Center for Supercomputing R & D 客員研究員．昭和 62 年 IFAC World Congress Young Author Prize，平成 9 年情処坂井記念特別賞，平成 20 年 LSI オブザイヤー準グランプリ，Intel Asia Academic Forum Best Research Award，平成 21 年 IEEE Computer Society Golden Core Member 等受賞．IEEE Computer Society 理事，情報処理学会 ARC 主査・会誌 HWG 主査・論文誌 HG 主査，文科省地球シミュレータ中間評価委員・次世代スーパーコンピュータプロジェクト中間評価作業部会専門委員・情報科学技術委員，経産省/NEDO “アドバンスト並列化コンパイラ”・“リアルタイム情報家電用マルチコア” プロジェクトリーダ，等歴任．