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. 3　96–106 (Sep. 2009)

マルチコア上でのOSCAR APIを用いた並列化コンパイラによる低消費電力化手法

間瀬正啓[†1]　中川亮[†1]　大國直人[†1]
白子準[†1]　木村啓二[†1]　笠原博徳[†1]

マルチコアプロセッサ上でアプリケーションプログラム実行時の消費電力を削減するためには，コンパイラやユーザによるプログラム実行中のチップ内リソースの適切な周波数・電圧・電源制御が必要であり，電力制御の指示が可能な API の利用が必要となる．本論文ではリアルタイム情報家電用マルチコア向けの OSCAR（Optimally Scheduled Advanced Multiprocessor）API を用いた自動並列化コンパイラによる低消費電力化手法を提案する．本手法は OSCAR コンパイラに実装されており，最小時間での処理を保証しつつ消費電力を最小化する最速実行モードと，メディアアプリケーションなどのリアルタイム処理において，リアルタイム制約を満たしつつ消費電力を最小化するリアルタイム制約モードをサポートする．NEDO の “リアルタイム情報家電用マルチコア技術” プロジェクトにおいて，ルネサステクノロジ，日立製作所，早稲田大学が開発した 8 コアの情報家電用マルチコア RP2 上で提案手法を評価したところ，4 コアを用いた場合の最速実行モードにおいて，消費エネルギーが SPEC2000 の art で 13.06%，SPEC2000 の equake で 3.99%，AAC エンコーダで 3.84%，MPEG2 デコーダで 9.01%削減されることが確かめられた．また，8 コアを用いた場合のリアルタイム制約モードにおいて，平均電力が AAC エンコーダで 87.9%，MPEG2 デコーダで 76.0%削減されることが確かめられた．

A Power Reduction Scheme for Parallelizing Compiler Using OSCAR API on Multicore Processors

Masayoshi Mase,[†1] Ryo Nakagawa,[†1]
Naoto Ohkuni,[†1] Jun Shirako,[†1] Keiji Kimura[†1]
and Hironori Kasahara[†1]

Efficient power reduction of an application program on multicore processors requires appropriate power control namely frequency-voltage control and power gating for on-chip resources by parallelizing compilers or users. These power reduction techniques need an application program interface (API) that enables power control inside a user program. This paper proposes a power reduction scheme for multicore processors using OSCAR API developed in NEDO “Multicore for Realtime Consumer Electronics” project. The proposed scheme has been implemented in OSCAR compiler to realize the power reduction for fastest execution mode, which minimizes power consumption without performance degradation, and the realtime execution mode to minimize power consumption under realtime constrains. The proposed scheme is evaluated on an 8 cores SH4A multicore processor RP2, newly developed for consumer electronics by Renesas Technology Corp., Hitachi, Ltd. and Waseda University in the above project. For the fastest execution mode, using 4 cores, consumed energy was reduced by 13.06% for SPEC2000 art, 3.99% for SPEC2000 equake, 3.84% for AAC encoder and 9.01% for MPEG2 decoder. Also, for the realtime execution mode, using 8 cores, consumed power was reduced by 87.9% for AAC encoder and 76.0% for MPEG2 decoder.

1. はじめに

半導体集積度向上にともなったスケーラブルな処理性能と高い電力効率を実現するために，マルチコアプロセッサは次世代プロセッサアーキテクチャの主流となりつつある．これらマルチコアプロセッサの有効利用には実行するプログラムの適切な並列化が必須であり，従来より多くの自動並列化コンパイラに関する研究が行われている[1)–3)]．OSCAR コンパイラ[4),5)] では，従来の市販コンパイラで用いられていたループ並列処理に加え，粗粒度タスク並列処理，近細粒度並列処理を組み合わせたマルチグレイン並列処理を実現しており，プログラム全域にわたる並列化が可能である．

また，マルチコアでは処理性能の向上に加え，増加する消費電力をいかに抑えるかが大きな課題となっており，従来から様々な手法が提案されている．たとえばキャッシュミス回数測定用カウンタや命令キューなどのハードウェアサポートにより実行時にプログラム中の各フェーズにおける負荷を判断し，不要なリソースを停止させる Adaptive Processing[6)] や，計算資源の各部分に対して実行時の負荷に応じた周波数・電圧制御（FV 制御）を行う Online Methods for Voltage and Frequency Control[7)] や実行時に各プロセッサの負荷の分散度を計算し，負荷不均衡が発生した場合には DVFS（Dynamic Voltage Frequency

†1 早稲田大学基幹理工学部情報理工学科
Department of Computer Science, Waseda University

Scaling）やタスク再配置を行うことで消費エネルギーの削減を行う手法[8] や，ループイタレーションレベルのプロセッサ間の負荷不均衡に対して，実行時のハードウェアサポートにより電力制御を行う Thrifty Barrier[9] などがある．しかし，これらの実行時の情報を利用した手法では，プログラム全域にわたるグローバルな消費電力の最適化は難しく，局所的な最適化にとどまってしまう．また，実行時電力制御のための処理やハードウェア追加が必要となるため遅延が大きいという欠点がある．このような手法に対して，コンパイル時の静的な情報に基づく低消費電力化手法では，プログラムの解析による詳細な情報を利用することができるため，よりきめ細かな電力制御が可能となり，プログラム全域にわたる消費電力の最適化が実現できる．コンパイラ制御によるシングルプロセッサの低消費電力化手法として compiler-directed DVS（dynamic voltage scaling）[10] があげられる．しかし，この手法はシングルプロセッサのみを対象としており，マルチコアプロセッサ向けの並列処理と電力制御との両立を実現できていない．マルチコアプロセッサ向けの OSCAR コンパイラではマルチグレイン並列処理による並列性抽出に加え，プログラムの各部分に対するコアごとの適切な動作周波数/電圧および電源遮断のタイミングを決定し，必要な処理性能を維持しつつ電力を低減させている[11]．

さらに，これらの OSCAR コンパイラによる並列化および低消費電力化を様々な種類のマルチコアに適用する目的で OSCAR API[12] が提案されている．OSCAR API は NEDO “リアルタイム情報家電用マルチコア技術” プロジェクトにおいて，早稲田大学，東芝，日本電気，パナソニック，日立製作所，富士通研究所，ルネサステクノロジによって開発された情報家電用マルチコア向け並列化 API である．OSCAR API を用いることで，OSCAR コンパイラによる最適化を様々なマルチコアで実現することができ，並列プログラムの大幅な生産性向上が期待できる．

本論文では，OSCAR コンパイラによって従来実現されてきた低消費電力化手法に加えて，3.2 節で述べる実行時のコスト変動によるプロセッサ間の負荷不均衡に対する低消費電力化手法，および 3.3 節で述べるリアルタイムアプリケーションを対象とした低消費電力化手法を提案する．さらに，これら OSCAR コンパイラによる低消費電力化手法を OSCAR API を用いて実現し，8 コアの情報家電用マルチコア RP2 上で評価を行った．コンパイラによるマルチコアの電力制御を実マルチコア上で実現し，評価を行ったのは，筆者らの知る限り本研究が世界で初めてである．

本論文の構成は以下のとおりである．2 章では提案する OSCAR コンパイラによる低消費電力化手法の前提となるマルチグレイン並列処理について述べる．3 章では OSCAR コンパイラによる低消費電力化手法について述べる．4 章では OSCAR API における電力制御用指示文について述べる．そして，5 章で性能評価について述べ，6 章でまとめを述べる．

2. OSCAR コンパイラによるマルチグレイン並列処理

本章では OSCAR コンパイラによるマルチグレイン並列処理の概要を述べる．

2.1 マクロタスク生成

マルチグレイン並列処理では，プログラムは基本ブロックおよびその融合ブロック BPA，ループなどの繰返しブロック RB，サブルーチン，関数を表す SB の 3 種類のマクロタスク MT[5] すなわち粗粒度タスクに分割される．**図 1** に示すように，RB や SB の内部は再帰的にマクロタスク分割され，階層的なマクロタスク構造となる．

2.2 マクロタスクグラフ生成

各階層におけるマクロタスク間のデータ依存，制御フローをもとに最早実行可能条件解析[4),5)] を適用し，マクロタスク間の並列性を抽出したマクロタスクグラフ MTG を生成する．マクロタスクグラフも入れ子構造をとり，プログラム全域にわたる階層的な粗粒度タスク並列性が抽出される．

2.3 マクロタスクスケジューリング

階層的に定義されたマクロタスクは，プロセッサの集合であるプロセッサグループ PG により実行される．適切な PG のグルーピングとタスクスケジューリングにより，階層的な並列性が有効利用される．この際，並列ループは複数の小ループに分割され，粗粒度タスクとしてプロセッサに割り当てられる．

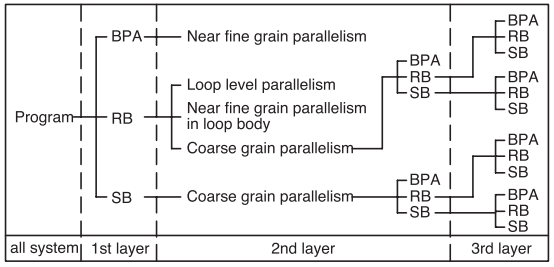

図 1　階層的マクロタスク定義
Fig. 1　Hierarchical macro task definition.

3. OSCAR コンパイラによる低消費電力化手法

マルチグレイン並列処理によりプログラムの最大限の並列化が可能であるが，実プログラムの完全な並列化は一般的に難しく，同期やデータ送受信によりプロセッサがアイドル状態となる可能性がある．また，リアルタイム処理においてタスク実行が早く終了した場合，デッドライン時刻までの待機時間が発生する．これらプログラム中のアイドル時間（余裕度）を利用しプロセッサの動作周波数低減や電源遮断を適用することにより，効果的な電力削減が可能となる．

本章では OSCAR コンパイラによる低消費電力化手法について述べる．本手法は以下のアーキテクチャサポートを持つマルチコア上で最適に利用可能である．

- プロセッサコアごとに周波数が可変
- 周波数に応じて電圧も低減可能
- コアごとの電源/クロック遮断が可能

3.1 コンパイル時における低消費電力化手法

本節では，コンパイル時の情報をもとに静的に適用される低消費電力化手法の概要を述べる．本手法はプログラムの最小処理時間を維持するための最速実行モード，与えられたデッドライン制約の範囲内における電力消費量最小化を目指すデッドライン制約モードの 2 つの実行モードを持つ．マクロタスクグラフに対する一般的な低消費電力化手法の手順は以下のとおりである．

(1)　マクロタスクグラフの処理時間の算出
(2)　各マクロタスクにおける動作周波数の決定
(3)　アイドル時間に対する電源遮断

3.1.1 マクロタスクグラフの処理時間

マルチグレイン並列処理により，マクロタスク MT は各プロセッサグループ PG に適切に割り当てられる．**図 2** に示す例では，3 つの PG に対し MT がスケジューリングされ，ユーザによって指定されたデッドライン時間が定められている．図中の time が時間経過を表しており，周波数が最速の場合の 1 クロックを単位時間とする．また，MT 間の実線はデータ依存を表している．当該 MTG の実行終了時間 T_{MTG} を表すため，以下の定義を行う．マクロタスク MT_i に対して，

T_i：MT_i の処理時間

T_{start_i}：MT_i の実行開始時刻

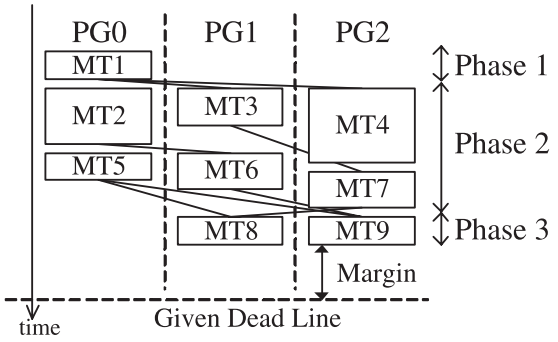

図 2　マクロタスクのスケジューリング結果の例
Fig. 2　Example of macro task scheduling result.

T_{end_i}：MT_i の実行終了時刻

を定義する．また，当該 MTG の開始時刻を 0 とする．図 2 の入口ノードである MT_1 のように，当該 PG が最初に実行する MT であり，かつ他の MT にデータ依存しない MT_i の実行開始時刻 T_{start_i} は

$$T_{start_i} = 0$$

となり，実行終了時刻は

$$T_{end_i} = T_{start_i} + T_i = T_i$$

となる．一方それ以外の MT_i に関しては，当該 PG が先行して実行するマクロタスク MT_j と MT_i がデータ依存するマクロタスク集合 $\{MT_k, MT_l, \ldots\}$ が終了した時点で MT_i の実行が開始されるため，実行開始時刻は

$$T_{start_i} = max(T_{end_j}, T_{end_k}, T_{end_l}, \ldots)$$

となり，終了時刻は

$$T_{end_i} = T_{start_i} + T_i$$

となる．図 2 を例として考えると MT_2，MT_3 は MT_1 が終了した時点で実行されるため開始時刻は

$T_{start_2} = T_{start_3} = T_{end_1} = T_1$ となり，終了時刻は

$$T_{end_2} = T_{start_2} + T_2 = T_1 + T_2,$$

$$T_{end_3} = T_{start_3} + T_3 = T_1 + T_3$$

である．また，MT_6 は MT_2 と MT_3 が終了した時点で実行が開始されるため

$T_{start_6} = max(T_{end_2}, T_{end_3})$

より，

$T_{start_6} = max(T_2, T_3) + T_1$

となる．同様に計算し，出口ノードである MT_8 の終了時刻は

$T_{end_8} = T_1 + T_8 + max(T_2 + T_5, T_6 + max(T_2, T_3), T_7 + max(T_3, T_4))$

と表される．

ここで一般形を考えると，出口ノード MT_{exit} の終了時刻は

$T_{end_{exit}} = T_m + T_n + \ldots + max_1(\ldots) + max_2(\ldots) + \ldots$

と表記される．入口ノードの実行開始時刻を 0 としたため，この $T_{end_{exit}}$ が当該 MTG の実行終了時間 T_{MTG} となる．

3.1.2　各マクロタスクにおける動作周波数の決定

MTG に対して与えられたデッドライン時間を $T_{MTG\_deadline}$，全 MT を最速の周波数で実行した場合の処理時間を T_{MTG_fast} と定義する．T_{MTG_fast} は T_{MTG} の式を用いて算出される．これらの定義より，2 つの実行モードにおける制約条件は以下のようになる．

- 最速実行モード：$T_{MTG} = T_{MTG\_fast}$
- デッドライン制約モード：$T_{MTG} \leq T_{MTG\_deadline}$

本手法では，それぞれの制約条件を満たしつつ全体の電力消費量が最小となるよう各 MT の動作周波数を決定する[11]．

3.1.3　アイドル時間に対する電源遮断

図 3 に示すように，各 MT の動作周波数決定後も同期待ちなどによりプロセッサがアイドル状態となる場合がある．ただし，図 3 中の MID は最速周波数の半分の動作周波数で当該 MT を実行することを表す．本手法ではアイドル状態にあるプロセッサに対して，電源遮断あるいはクロック遮断を適用することでアイドル中の無駄な電力消費を削減する[11]．

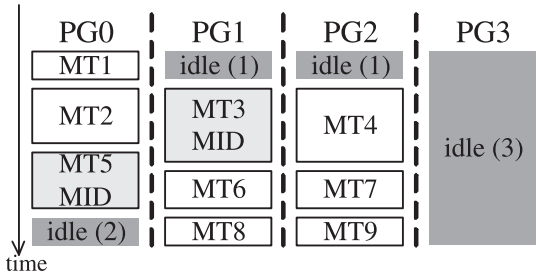

図 3　周波数・電圧制御の結果の例
Fig. 3　Example of F/V control result.

3.2　実行時負荷不均衡に対する低消費電力化手法

本節では，実行時に生じうるプロセッサ間の負荷不均衡に対する低消費電力化手法を提案する．コンパイル時に各プロセッサに処理が均等に割り当てられたとしても，プログラムの入力などに依存する処理の不確定性により，実行時にプロセッサ間で負荷の不均衡が起こる可能性がある．特に，ループボディに条件分岐を含む並列ループを分割し各プロセッサに割り当てた場合においてそのような負荷不均衡は発生しやすく，提案手法ではそのような並列ループをコンパイル時に検出し，電力制御命令をプログラム中に挿入することで，実行時の負荷不均衡により生じるアイドル時間に対して電源遮断を行い，無駄な電力消費を削減する．

あるループを本手法の制御対象とする必要条件は以下のとおりである．

- 複数のプロセッサによって並列処理される．
- ループボディに条件分岐が存在する，または回転数が変動する内部ループを持つ．
- 電力制御オーバヘッドと比較して処理時間が十分に大きい．

上記の条件を満たす実行時負荷不均衡ループを含む MTG 中において，負荷不均衡ループを実行した後に，プロセッサ間でバリア処理が必要な場合にビジーウェイトによって消費される電力を削減するため，先行してバリアに到達したプロセッサに対して電源遮断を適用する．ただし，「電力制御オーバヘッドと比較して処理時間が十分に大きい」とは，並列ループの逐次実行時のコンパイラによる推定処理時間を Cost，当該並列ループを処理するプロセッサ数を PE，電源遮断状態から最速の電力状態に復帰するためのオーバヘッドを $Overhead_{poweroff}$，電力制御オーバヘッドに対するループのコスト比を Ratio としたとき，以下の式を満たすことをいう．

$Overhead_{poweroff} \times Ratio \leq Cost/PE$

なお，5 章の RP2 上での評価においては $Overhead_{poweroff}$ には実測値の 100 [μs] を用い，Ratio は 100 とした．

提案手法では，カウンティングセマフォアを用いることで最後にバリアに到達した PG を実行時に判定し，以下の処理に移る．

- 最終 PG：電源遮断中の他 PG を復帰．
- 他 PG：電源遮断状態に移行．

図 4 に負荷不均衡のある並列ループに対する本手法の適用例を示す．この例では，負荷

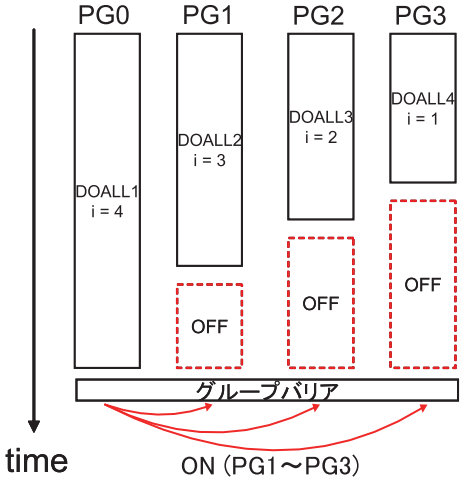

図 4　負荷不均衡のある並列ループに対する低消費電力化手法の適用例
Fig. 4　Example of applying power reduction scheme for parallel loop with load imbalance.

不均衡により PG3，PG2，PG1 の順にバリアに到達し電源遮断状態に移行している．最後にバリアに到達した PG0 により PG1〜PG3 に対して電源復帰命令が発行される．

3.3　デッドライン時刻までの待機電力最小化

3.1 節ではデッドライン制約に応じて各 MT の処理周波数を適切に低減させる手法について述べた．しかしデッドライン時間が十分に大きい場合，全 MT を最低周波数に指定した場合でもデッドライン時刻よりも早く処理が終了することがある．特にメディアアプリケーションなどの周期的なリアルタイム処理においては，全 MT の処理終了後にプログラムへの入力待ちなどによる待機時間が発生する可能性がある．本節では，このような制約時刻までの待機をともなうデッドライン制約モードをリアルタイム制約モードと定義し，待機時間中の不要な電力消費を最小化する手法を提案する．本手法は 4 章で述べるプログラム中での現在時刻を取得するタイマ API を用いる．

3.3.1　制御対象階層の検出

提案手法では，ユーザにより与えられたデッドライン時間と 3.1.1 項の方法によってコンパイル時に推定した周波数制御後の MTG の処理時間を比較し，当該 MTG における待機時間 T_{wait} を算出する．なお，周波数制御後の各 MT の処理時間は，その周波数に応じた比率（50%の周波数なら 2 倍，25%の周波数なら 4 倍）を MT の処理時間にかけて推定する．動作周波数を低減させてもバスやメモリの速度は変わらないため，動作周波数を半分にしても実際には処理時間は 2 倍にならないこともあるが，本手法では最大の遅延を考慮することで，デッドライン時刻を超過するのを防いでいる．

また，当該 MTG に割り当てられた全プロセッサが電源遮断と動作状態への復帰を行うためのオーバヘッドを T_{OH_power}，同様にクロックの遮断・復帰に必要なオーバヘッドを T_{OH_clock} とする．これら各 MT の処理時間および電源遮断，クロック遮断オーバヘッドの推定では，対象マルチコア上でのプロファイル情報を用いることで精度を高めている[11]．提案手法ではこれらのパラメータを比較し，以下の判断基準で電力制御を適用する．

- $T_{wait} \leq T_{OH\_clock}$：制御なし
- $T_{OH\_clock} < T_{wait} \leq T_{OH\_power}$：クロック遮断
- $T_{OH\_power} < T_{wait}$：電源遮断

3.3.2　タイマを用いた実行時デッドライン管理

当該 MTG が電源遮断あるいはクロック遮断の適用階層と判断された場合，その MTG での最後の MT を処理するプロセッサ（PG 内の先頭プロセッサ）がマスタプロセッサに指定される．マスタプロセッサ以外のプロセッサは，自身に割り当てられた最後の MT の終了後に電源遮断あるいはクロック遮断状態に移行する．マスタプロセッサは自身に割り当てられた先頭 MT の開始時刻をタイマを用いて取得し，最後の MT の終了時に最低動作周波数に移行後，タイマ API を用いてデッドラインまで待機する．デッドラインに到達後，電源遮断あるいはクロック遮断状態にある他のプロセッサを復帰させ，次の処理に移る．

図 5 に提案手法の適用例を示す．ここではコア 0（PG0 内先頭プロセッサ）がマスタプロセッサに選択され，最後の MT6 終了後にデッドライン時刻まで最低の消費電力状態で待機している．

4.　OSCAR API における電力制御用指示文

本章では提案手法を実装した OSCAR コンパイラが出力する OSCAR API における電力制御用の指示文について述べる．OSCAR API は OpenMP をベースとした API で，並列実行，メモリ配置，データ転送，電力制御，タイマ，同期の 6 つのカテゴリ，15 の指示文で構成される[12]．本論文では OSCAR API のうち，**図 6** に示す 3 つの API を用いて，情報家電用マルチコア RP2 上での電力制御を実現した．

指示文 get_fvstatus によって，pe_no 番目のプロセッサの，module_id で示されたモジュールの電力状態を変数 fv_state に取得することができる．モジュールとしては，CPU，キャッ

```
#pragma oscar get_fvstatus(pe_no, module_id, fv_state)
#pragma oscar fvcontrol(pe_no, (module_id, fv_state))
#pragma oscar get_current_time(current_time, timer_no)
```

図 6　本手法で用いた OSCAR API
Fig. 6　OSCAR API for proposed scheme.

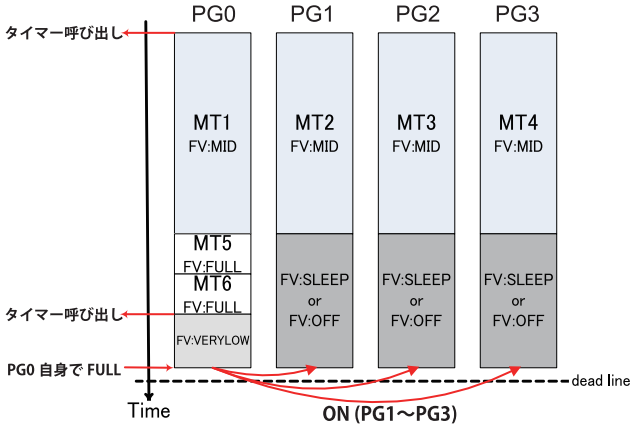

図 5　リアルタイム制約モードの適用例
Fig. 5　Example of applying realtime execution mode.

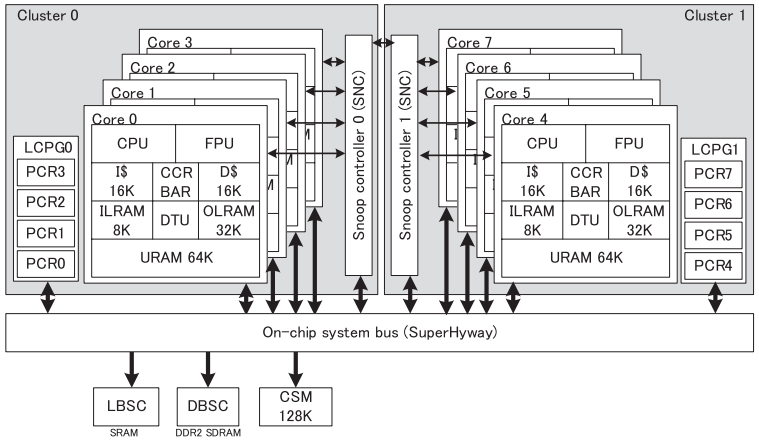

図 7　RP2 の構成図
Fig. 7　Configuration of RP2.

シュあるいは CPU とキャッシュを含むコア全体など，マルチコアを構成する各種要素を指定できる．本手法ではこの指示文を用いて，状態遷移中の電源復帰命令発行の防止を実現した．また，指示文 fvcontrol によって，pe_no 番目のプロセッサの，module_id で示されたモジュールの電力状態を変数 fv_state で指定した電力状態に変更することができる．また，指示文 get_current_time によって，timer_no 番目のタイマーから現在時刻を取得し，変数 current_time に取得することができる．この指示文を用いて，3.3 節で述べたリアルタイム制約モードでの低消費電力化手法を情報家電用マルチコア RP2 上で実現した．

5. 性能評価

3 章で述べた低消費電力化手法を実装した OSCAR コンパイラの性能評価を情報家電用マルチコア RP2 上で行った．

5.1 情報家電用マルチコア RP2

本節では，提案手法の評価に用いた RP2 について述べる．RP2 は NEDO の “リアルタイム情報家電用マルチコア技術” プロジェクトにおいて，早稲田大学，ルネサステクノロジ，日立製作所によって共同開発された情報家電向けマルチコアプロセッサで，1 つのチップ上に 8 つの SH-4A コアを搭載している．RP2 の構成図を**図 7** に示す．各プロセッサコアには，CPU，キャッシュ，ローカルメモリ（命令メモリ ILRAM，データメモリ OLRAM），分散共有メモリ（URAM），データ転送ユニット（DTU）が搭載されており，4 コアまではスヌープコントローラが MESI プロトコルでキャッシュコヒーレンシを保証する SMP モードとして動作する．5 コア以上を使用する場合は，ソフトウェアが明示的にキャッシュコヒーレンシを保証する必要がある．

RP2 では，各 CPU の周波数を 600 MHz，300 MHz，150 MHz，75 MHz に独立して変更することが可能である．また，供給電圧はチップ一括での変更が可能である．それに加え

表 1　RP2 の電力状態
Table 1　Power Modes of RP2.

状態名	クロック遮断対象モジュール	電源遮断対象モジュール	消費電力［W］
FULL（600 MHz，1.404 V）	なし	なし	5.29
MID（300 MHz，1.196 V）	なし	なし	2.44
LOW（150 MHz，1.004 V）	なし	なし	1.22
VERYLOW（75 MHz，1.004 V）	なし	なし	0.99
Light Sleep（1.004 V）	CPU	なし	1.096
Normal Sleep（1.004 V）	CPU，cache，ILRAM，OLRAM	なし	0.743
Resume Standby（1.004 V）	URAM	CPU，cache，ILRAM，OLRAM，DTU	0.566
CPU off（1.004 V）	なし	CPU，cache，ILRAM，OLRAM，DTU，URAM	0.554

て，コア中の CPU のみのクロックを遮断する Light Sleep，コア中の URAM と DTU 以外のクロックを遮断する Normal Sleep，コア中の URAM のクロックを遮断し，それ以外を電源遮断する Resume Standby，コアの電源をすべて遮断する CPU off の 4 つの低電力状態もとることが可能である．**表 1** に RP2 の持つ電力状態と，8 つのプロセッサコアすべてを各電力状態にした場合の消費電力を示す．

また，電源復帰は他のコアからの割込み処理によって実現されている．

5.2　最速実行モードの評価

最速実行モードは 3.1 節で述べたコンパイル時における低消費電力化手法と 3.2 節で述べた実行時負荷不均衡に対する低消費電力化手法を用いており，本節ではこれらの手法をまとめて本手法と呼ぶ．

SPEC2000 の art，equake および MediaBench の MPEG2 デコーダを制約付き C [13] に書き換えたプログラムとルネサステクノロジ提供の AAC エンコーダを用いて，RP2 上で最速実行モードでの電力評価を行った．なお，MPEG2 デコーダの実行条件として，画面サイズは 352 × 240，フレームレートは 30 [fps]，ビットレートは 5000 [kbps] のものを用いた．AAC エンコーダについては，16 bit ステレオ，ビットレート 128 [kbps] のものを用いた．評価には 4 コアの SMP モードを用い，使用しないプロセッサはプログラム開始時に CPU off を適用した．また，電源制御には Resume Standby を使用した．

4 コアでの並列化による速度向上としては，1 コアを用いた場合と比較して，art において 2.49 倍，equake において 2.67 倍，AAC エンコーダにおいて 3.29 倍，MPEG2 デコーダにおいて 2.67 倍の速度向上が得られた．

本手法を適用した場合と適用しない場合の，実行時間と RP2 チップ全体の消費エネルギーを比較した．ただし，データの入出力部分は評価から除いている．まず実行時間を**図 8** に

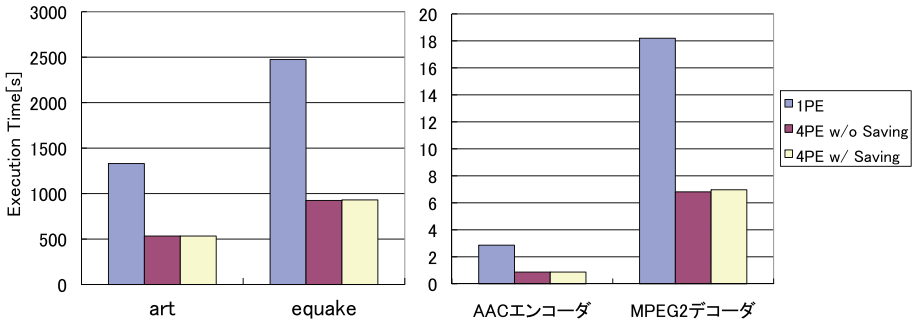

図 8　最速実行モードにおける実行時間
Fig. 8　Execution time in fastest execution mode.

示す．図の横軸が各アプリケーション，縦軸が実行時間を表しており，各アプリケーションのバーは左から，1 コアを用いた場合，4 コアを用いて本手法を適用しなかった場合，4 コアを用いて本手法を適用した場合を表している．4 コアを用いた場合における本手法適用による遅延率は art において 0.0042%，equake において 0.62%，AAC エンコーダにおいて 0.11%，MPEG2 デコーダにおいて 2.22%であった．これらの遅延は 3.2 節で述べた実行時負荷不均衡に対する低消費電力化手法によるものである．

次に消費エネルギーを**図 9** に示す．本手法による消費エネルギー削減効果は art において 13.05%（1795.32 [J] から 1560.92 [J]），equake において 3.99%（3071.64 [J] から 2949.07 [J]），AAC エンコーダにおいて 3.84%（3.03 [J] から 2.91 [J]），MPEG2 デコーダにおいて 9.01%（25.68 [J] から 23.37 [J]）となった．

ここで最速実行モードにおいて，3.2 節で述べた実行時負荷不均衡並列ループに対する制

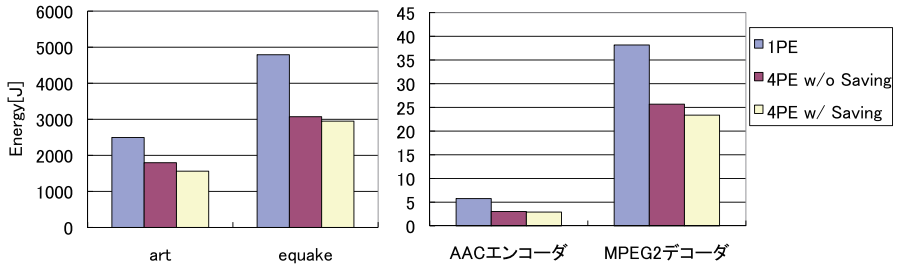

図 9　最速実行モードにおける消費エネルギー
Fig. 9　Energy consumption in fastest execution mode.

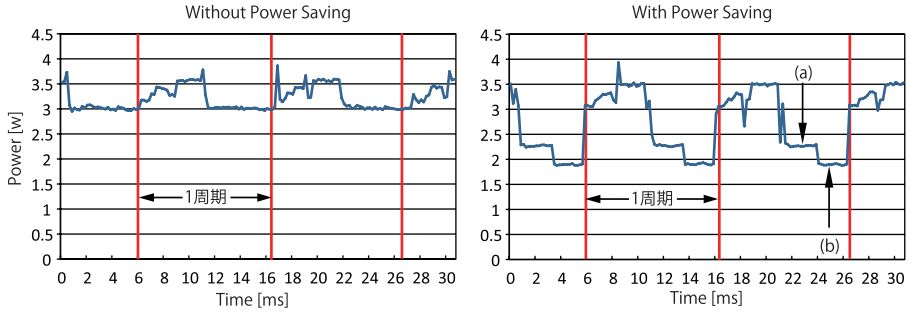

図 10　art の電力波形
Fig. 10　Power waveform for art.

御が特に有効であった art について詳しく考察する．art の処理時間の約 6 割を占める並列ループにおいて，最も処理が早く終了したプロセッサと最も遅く処理が終了したプロセッサでは，平均で 51.1%の時間差があった．この並列ループは，コンパイル時には均等に分割され各プロセッサに処理が割り当てられたが，プログラムの入力によって並列ループの内側のループの回転数が決定されるため，実行時にプロセッサ間で負荷不均衡が生じた．本手法ではこのループを含む MTG に対して電力制御が適用された．

art を実行した際の電力波形の抜粋を**図 10** に示す．図の横軸が時刻，縦軸が消費電力を表している．また，縦の破線に囲まれた領域がループの 1 周期を表している．図 10 で電力が低減されていることには，前述したループを含む MTG に対する電力制御が寄与している．図 10 の電力制御が適用された箇所の波形を見ると，まず 2.25 [W] 付近まで電力が下がり（a），次に 1.9 [W] 付近まで電力が下がっていることが分かる（b）．これは，処理時間の約 6 割を占めるループを実行する際，各プロセッサが処理終了後にバリアに到達した順に電力制御が適用されたためである．

次に，本手法で発生した遅延率について考察する．本手法では，対象 MTG のバリア同期の直前のタイミングで電源制御を適用していたため，制御オーバヘッドに応じた遅延が生じていた．最も遅延率が大きかった MPEG2 デコーダにおいては，9.01%の消費エネルギーの削減効果が得られた一方，遅延率が 2.22%となっている．最速実行モードにおいては遅延が発生しないことが理想となるため，遅延率を低減させる必要があると考えられる．遅延を生じさせずに電力制御を行うためには，制御オーバヘッドを考慮して電源制御をバリア同期よりも一定時間前のタイミングで行う必要があり，このような制御の最適化は今後の課題である．

5.3　リアルタイム制約モードの評価

リアルタイム制約モードは 3.1 節で述べたコンパイル時における低消費電力化手法と 3.2 節で述べた実行時負荷不均衡に対する低消費電力化手法と 3.3 節で述べたデッドライン時刻までの待機電力最小化手法を用いており，周期的なリアルタイムデッドラインを持つようなプログラムを対象としたモードである．本節ではこれらの手法をまとめて本手法と呼ぶ．

ルネサステクノロジ提供の AAC エンコーダと，MediaBench の MPEG2 デコーダを制約付き C に書き換えたプログラムを用いて，RP2 上でリアルタイム制約モードでの電力評価を行った．AAC エンコーダにおけるリアルタイム制約は 44.1[フレーム/s]，MPEG2 デコーダにおけるリアルタイム制約は 30[フレーム/s] とした．なお，評価に先立ち，RP2 上でこれらのプログラムを実行したプロファイル情報（プログラム中のループや関数の実行時間や実行回数の情報）をコンパイラに与え，各マクロタスクのコスト見積り精度を高めている．

本手法を適用した場合と適用しない場合の平均電力を**図 11** に示す．図 11 の各アプリケーションの左側の 2 つのバーは，リアルタイム制約を満たせる最少のコア数で実行した際の，本手法適用時と非適用時の平均電力を示している．AAC エンコーダにおいては 1 コア，MPEG2 デコーダにおいては 2 コアでリアルタイム制約を満たすことができた．右側の 2 つのバーは 8 コアを用いて実行した際の，本手法適用時と非適用時の平均電力を示している．各コア数における本手法による電力削減効果は，AAC エンコーダにおいて 1 コア使用時に 66.0%（1.88 [W] から 0.64 [W]），8 コア使用時に 87.9%（5.40 [W] から 0.65 [W]）であり，MPEG2 デコーダにおいて 2 コア使用時に 20.1%（2.44 [W] から 1.95 [W]），8 コア

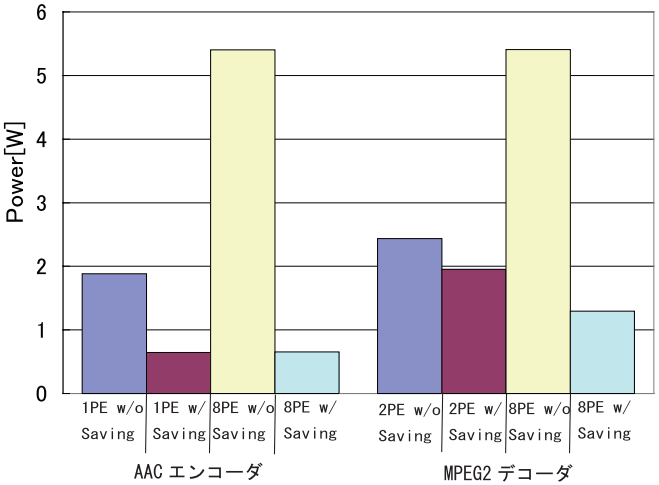

図 11　リアルタイム制約モードでの平均電力
Fig. 11　Average power in realtime execution mode.

使用時に 76.0%（5.41 [W] から 1.30 [W]）であった．

AAC エンコーダおよび MPEG2 デコーダを 8 コアを用いて実行した際の電力波形の抜粋をそれぞれ**図 12** と**図 13** に示す．図の横軸が時刻，縦軸が電力を表す．AAC エンコーダにおいては，リアルタイムデッドラインまでの余裕度が十分に大きいため，エンコード処理は LOW の状態で行われ，その後デッドラインまで VERY LOW および電源遮断（Resume Standby）が適用された．MPEG2 デコーダにおいては，リアルタイムデッドラインまでの余裕度が小さいため，主要のデコード処理については LOW の状態で 8 コアで並列処理されたが（a），他の処理については FULL の状態と電源遮断（Resume Standby）が併用して適用された．

リアルタイム制約を満たせる最少のコア数を用いて本手法を適用した場合と，8 コアを用いて本手法を適用した場合の平均電力を比較する．AAC エンコーダにおいては，1 コアで実行した場合でもデッドラインまでの余裕度が十分にあるため，8 コアを用いた場合と同様にエンコード処理が LOW の状態で実行された．8 コアを用いた場合，同期や電源復帰のオーバヘッドが 1 コアを用いた場合よりも大きいため，1 コアを用いた場合の方が 8 コアを用いた場合よりも 1.4%平均電力が低くなった．一方，MPEG2 デコーダにおいてはデッドラインまでの余裕度が小さいため，1 コアでのリアルタイム実行は不可能で，2 コアを用いた場合でも，主要のデコード処理を最速（FULL）でない周波数状態で実行するとリアルタイム制約が満たせなくなってしまう．そのため，2 コアを用いた場合は主要のデコード処理を FULL で実行後，デッドラインまで VERY LOW および電源遮断（Resume Standby）が適用された．そのため，8 コアを用いた場合の方が 2 コアを用いた場合よりも 33.7%平均電力が低くなった．

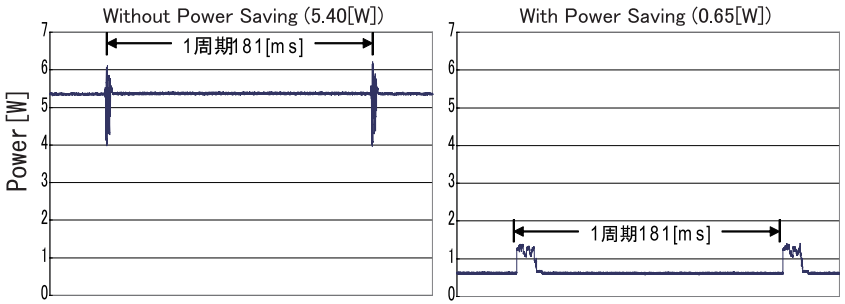

図 12　AAC エンコーダにおける電力波形
Fig. 12　Power waveform for AAC encoder.

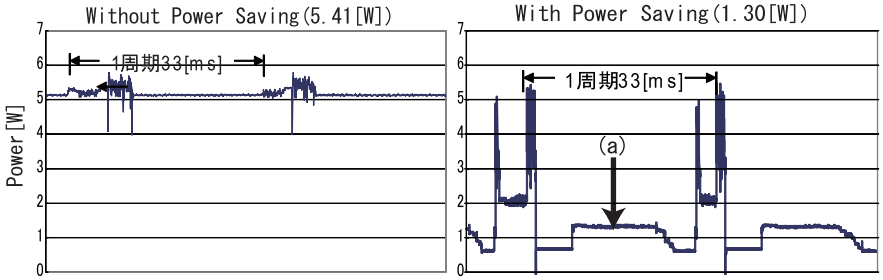

図 13　MPEG2 デコーダにおける電力波形
Fig. 13　Power waveform for MPEG2 decoder.

6. ま と め

本論文では OSCAR API を用いたマルチコアプロセッサ向けの低消費電力化手法を提案し，情報家電用マルチコア RP2 上で評価を行った．最速実行モードでは，実行時に負荷不

均衡が起こりうる並列ループを含む MTG において電源遮断を適用することで，ビジーウェイトなどによる無駄な電力消費の削減を実現し，SPEC2000 の art，equake および AAC エンコーダ，MPEG2 デコーダにおいて消費エネルギーをそれぞれ 13.06%，3.99%，3.84%，9.01%削減することができた．また，メディアアプリケーションのリアルタイム処理向けのリアルタイム制約モードでは AAC エンコーダと MPEG2 デコーダにおいて，リアルタイム制約を守りつつ平均電力をそれぞれ 87.9%，76.0%削減することができた．

謝辞　本研究の一部は NEDO“リアルタイム情報家電用マルチコア技術”“情報家電用ヘテロジニアスマルチコア技術”プロジェクト，早稲田大学グローバル COE“アンビエント SoC”の支援により行われた．

参　考　文　献

1) Wolfe, M.: *High Performance Compilers for Parallel Computing*, Addison-Wesley Publishing Company (1996).
2) Eigenmann, R., Hoeflinger, J. and Padua, D.: On the Automatic Parallelization of the Perfect Benchmarks, *IEEE Trans. Parallel and Distributed Systems*, Vol.9, No.1 (1998).
3) Hall, M.W., Anderson, J.M., Amarasinghe, S.P., Murphy, B.R., Liao, S., Bugnion, E. and Lam, M.S.: Maximizing Multiprocessor Performance with the SUIF Compiler, *IEEE Computer* (1996).
4) 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手法，電子情報通信学会論文誌，Vol.J73-D-1, No.12, pp.951–960 (1990).
5) 笠原：最先端の自動並列化コンパイラ技術，情報処理，Vol.44, No.4, pp.384–392 (2003).
6) Albonesi, D.H., et al.: Dynamically tuning processor resources with adaptive processing, *IEEE Computer* (2003).
7) Wu, Q., Juang, P., Martonosi, M. and Clark, D.W.: Formal Online Methods for Voltage/Frequency Control in Multiple Clock Domain Microprocessors, *11th International Conference on Architectural Support for Programming Languages and Operating Systems* (2004).
8) 高務，小林，小野寺：エネルギ最小周波数を利用したタスク再配置によるマルチプロセッサ向け消費エネルギ削減手法，電子情報通信学会技術研究報告，Vol.104, No.711, pp.37–42 (2005).
9) Li, J., Martinez, J.F. and Huang, M.C.: The Thrifty Barrier: Energy-Aware Synchronization in Shared-Memory Multiprocessors, *Proc. High Performance Computer Architecture* (2004).
10) Hsu, C. and Kremer, U.: The Design, Implementation and Evaluation of a Compiler Algorithm for CPU Energy Reduction, *The ACM SIGPLAN Conference on Programming Language Design and Implementation* (2003).
11) 白子，吉田，押山，和田，中野，鹿野，木村，笠原：マルチコアプロセッサにおけるコンパイラ制御低消費電力化手法，情報処理学会論文誌：コンピューティングシステム，Vol.47, No.ACS15 (2006).
12) OSCAR API 仕様書．http://www.kasahara.cs.waseda.ac.jp/api/regist.html
13) 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，組込みシステムシンポジウム 2007 (2007).

(平成 21 年 1 月 27 日受付)
(平成 21 年 5 月 1 日採録)

間瀬　正啓（学生会員）
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，現在に至る．

中川　　亮
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 21 年同大学大学院修士課程修了．平成 21 年株式会社 NTT データ入社，現在に至る．

大國　直人
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 21 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進学，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 年学振特別研究員 PD．平成 20 年アメリカ合衆国 Rice 大学コンピュータサイエンス工学科ポスドク研究員，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 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 年同教授，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〜2 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 年情報処理学会坂井記念特別賞，平成 20 年 LSI オブザイヤー準グランプリ受賞．IEEE Computer Society 理事，情報処理学会 ARC 主査，論文誌 HG 主査，文部科学省地球シミュレータ中間評価委員，経済産業省/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” 等プロジェクトリーダ歴任．