卒業論文

題目

ハードウェアスケジューラを用いた
省電力リアルタイムマルチコアプロセッサの改良

指導教員

佐々木 敬泰 助教

2015年

三重大学 工学部 情報工学科
計算機アーキテクチャ研究室

嶋谷 知(411822)

内容梗概

近年，組込みシステムの高機能化に伴い消費電力の増大が問題となっている．電力面で厳しい制約を持つ組み込みシステムでは高機能化によってより高い省電力性能が求められている．しかし，組込みシステムにはリアルタイム性を保証するため高性能なハードウェアを搭載する必要がある．そのため，本研究では高機能かつ省電力化可能なアプローチである単一命令セットアーキテクチャ(ISA：Instruction Set Architecture) ヘテロジニアスマルチコアプロセッサ (HMP：Heterogeneous Multi-core Processor) 構成を用いることでリアルタイム性を保証しつつ省電力化を行う．

単一 ISAHMP は同命令セットで異性能のコアを組み合わせたマルチコア構成となっており，タスク毎の負荷に合わせてコアを切り替えることで高い電力効率を実現する．商用製品への搭載例として ARM の big.LITTLE があり，省電力アプローチとしての有用性が証明されている．しかしながら，単一 ISAHMP のコア切り替えがタスクの処理完了時間に及ぼす影響が問題となっている．リアルタイム処理向け OS(RTOS：Real-Time OS) の提供するソフトウェアでのタスクスケジューリングでは正確な時間情報管理が困難となるため，既存の単一 ISAHMP を搭載したシステムはリアルタイム処理向けに最適化されていない．

そのため，先行研究ではハードウェアスケジューラを提案した．このハードウェアスケジューラの手法は，ローエンドコアで実行して最悪実行時間 (WCET：Worst Case Execution Time) がデッドラインを越えた場合にハイエンドコアに切り替えるというものである．

しかし，先行研究のスケジューリング手法は一度ハイエンドコアに切り替えたらデッドラインが終わるまでローエンドコアを使用しないため，ハイエンドコアで実行した場合に高速で処理して早期に終了してしまう可能性がある．早期に終了すると無駄な待ち時間が長くなるため，ローエンドコアでその待ち時間を利用すると省電力化が見込める．そのため，本研究ではローエンドコアの使用率を上げて省電力化を行うためにスケジューリング手法の改良を行った．

本研究で提案したスケジューリング手法は先行研究と比較して平均約 7%の消費電力の削減に成功した．

Abstract

Recently, an increasing in power consumption due to the high functionality of the embedded system become a problem. High functional embedded systems demand power saving performance with strict constraints. However, the embedded system need to be equipped with high-performance hardware to ensure real-time performance. Therefore, this paper propose the method to improve to do power saving while ensuring the real-time by using a single-instruction set architecture (ISA) heterogeneous multi-core processor (HMP) that is highly functional and power saving possible approaches.

Single ISA HMP is a multi-core processor that combines the core of a different performance with the same instruction set, to achieve a high power efficiency by switching the core according to the load of each task. The ARM's big.LITTLE as example of mounting the commercial products has been demonstrated that is useful as a power saving approach. However, the effect of core switching single ISA HMP is the problem on the processing completion time of the task. Since the accurate time information managed by the task scheduling in the software provided by the OS for real-time processing becomes difficult, the system with the existing single ISAHMP isn't optimized for real-time processing.

Therefore, Previous paper proposed a hardware scheduler. The hardware scheduler using low-end core switch to high-end core if worst case execution time (WCET) exceeds the deadline.

However, scheduling method of previous paper has a possibility that ended early at a high speed when running on high-end core because it does not use the low-end core until the end of the deadline after switching to high-end core. Since the early ends and useless latency becomes longer, if the waiting time used, using the low-end core can be expected power saving. Therefore, This paper propose to improve scheduling method in order to increase the utilization rate of the low-end core and perform the power saving. This paper's proposed scheduling method successed reducing the average about 7% of the power consumption that compared to previous paper.

目 次

1 はじめに **1**
1.1 研究背景 1
1.2 研究目的 1

2 マルチプロセッサ用リアルタイム OS **4**
2.1 リアルタイム OS 4
2.2 リアルタイム 5
2.3 ユニプロセッサ用 RTOS とマルチプロセッサ用 RTOS . . . 7
2.3.1 ユニプロセッサ 7
2.3.2 マルチプロセッサ 7
2.4 HMP 9
2.4.1 HMP における命令セット 9
2.4.2 単一 ISAHMP 上での RTOS 10

3 関連研究 **11**

4 スケジューリング手法 **12**
4.1 タスクスケジューリング 12
4.2 先行研究の手法 13
4.3 提案手法 16
4.4 提案手法のハードウェア化 18

5 性能評価 **20**
5.1 評価環境 20
5.2 ベンチマークプログラム 21
5.3 評価結果 24
5.4 考察 29

6 おわりに **30**

謝辞 **31**

参考文献 **31**

図 目 次

1.1 ヘテロジニアスマルチコア構成 2
2.2 ホモジニアス構成とヘテロジニアスマルチコア構成のプロセッサ 8
2.3 通常のスケジューリング手法 10
4.4 先行研究のスケジューリング手法 13
4.5 チェックポイント 14
4.6 動的スケジューリングアルゴリズム 15
4.7 提案アルゴリズム 17
4.8 Dynamic Scheduling Unit 19
4.9 Dynamic Scheduler 20
5.10 消費電力 26
5.11 デッドライン残量 26
5.12 ローエンドコア使用率 27
5.13 消費電力 (WCET=Deadline) 27
5.14 デッドライン残量 (WCET=Deadline) 28
5.15 ローエンドコア使用率 (WCET=Deadline) 28

表 目 次

5.1 評価環境 21
5.2 評価条件 23

1　はじめに

1.1　研究背景

近年，スマートフォンや自動車の制御システムなどの組み込みシステムの高機能化に伴いプロセッサのマルチコア化が進んでいる．しかし，組み込み分野では消費電力の制約を持つシステムが多く，高性能化に伴ってシステムの消費電力の増加が問題となる．

本研究では自動車やスマートフォンなどの組み込みシステムにおいて信頼性として要求されるものの一つであるリアルタイム性を持つシステムの消費電力に着目し，マルチコアの効率的な利用により省電力性能の向上を行う．

1.2　研究目的

現在の組み込み分野において単一 ISA のヘテロジニアスマルチコアプロセッサ (Heterogeneous Multi core Processor: HMP) が省電力化の手法として注目されている．これは，図 1.1 のようにハイエンドコアとローエンドコアの 2 種類のコアを用いることでタスクの負荷によって最適な使用コアに変更することができ，消費電力の削減が見込めるためである．また，単一 ISA にすることで異種のコアでも同じ命令が使用できるためコ

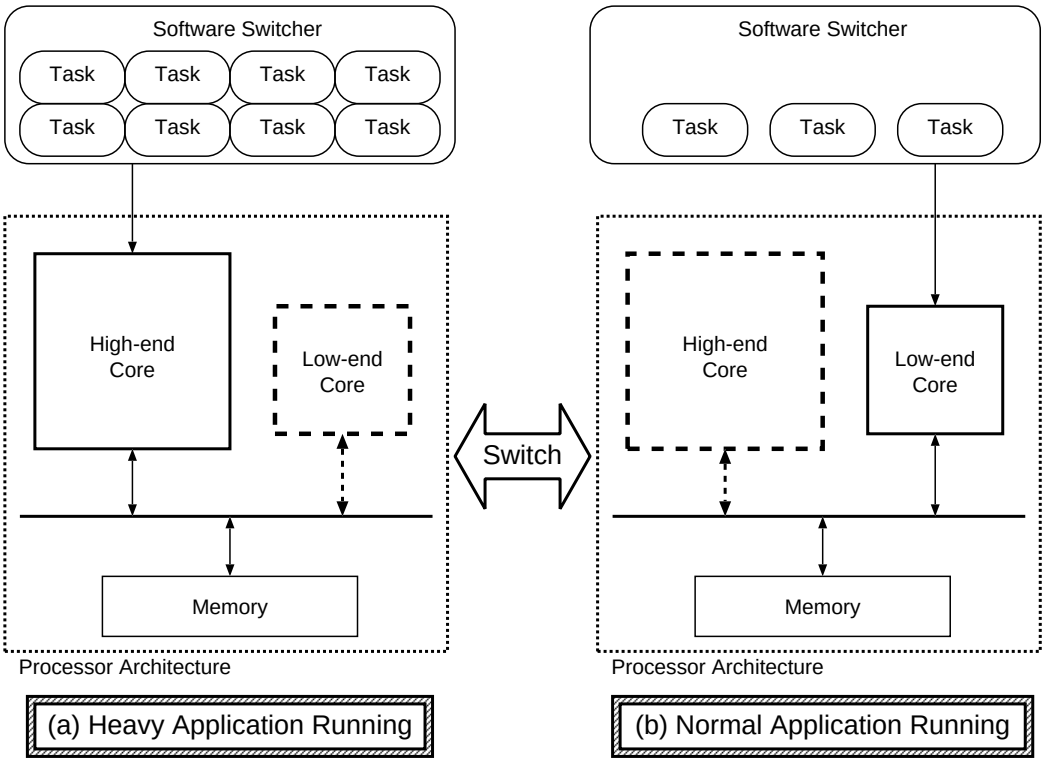

図 1.1: ヘテロジニアスマルチコア構成

ア切り替えが高速かつ容易になる．HMP 構成は ARM の big.LITTLE[2] で商用として採用されており，評価結果 [3] において省電力化に成功しているため有用性が高いと言える．しかしコア切り替えをソフトウェアで実装した場合，タスクのコンテキストスイッチによるオーバヘッドが問題となり実行時間の予測が困難となる．そのためタスクスケジューリングが困難となるためリアルタイム処理においては単一 ISAHMP 構成のプロセッサが最適化されていない．そこで先行研究 [1] では単一 ISAHMP 用

のスケジューリング手法とそのハードウェア化の提案を行った．しかし，先行研究のスケジューリング手法はタスク負荷が大きくなるとハイエンドコアの使用率が高くなり消費電力が大きくなる．スケジューリング手法の改良によってローエンドコアの使用率を上げるようにすることで消費電力の削減が可能であったため改良し，その評価を行った．

改良する対象のアルゴリズムは瀬戸らが行った単一 ISAHMP における動的スケジューリングアルゴリズム [1] である．この手法は，まず前処理として実行するリアルタイムタスクを処理時間単位で分割する．そしてそれぞれの分割点に WCET と実行命令数を記録するチェックポイントを設ける．その後ローエンドコアを用いて処理を行い，タスク実行中にタスクの実行状態を監視し WCET がデッドラインを越えると判定した場合にローエンドコアからハイエンドコアにコア切り替えを行うものである．

先行研究でのコア切り替えのアルゴリズムはローエンドコアからハイエンドコアへ遷移するのみであったため，ハイエンドコアに切り替えてから時間の余裕ができた場合無駄な待ち時間が発生する可能性がある．リアルタイム処理はデッドライン以内に処理を終えることができればよいため，待ち時間をローエンドコアで有効利用することが可能である．そのため更なる切り替え手法を提案する必要がある．また，先行研究では空

ループや明示的なアクセスで実装された単純なプログラムでスケジューリングアルゴリズムの検証を行っていたため実際的なタスクは考慮されていない．そのため，スケジューリングアルゴリズムの評価には先行研究とは異なるベンチマークプログラムを用いる．

本研究では新たなベンチマークを用いてスケジューリングアルゴリズムについての性能を評価し，更なる改善案を提案する．

2　マルチプロセッサ用リアルタイム OS

提案手法の説明をするのに先立ち，リアルタイム OS と単一 ISAHMP の関係の説明を行う必要があるため，本章ではリアルタイム性やマルチプロセッサについて概括する．

2.1　リアルタイム OS

リアルタイム OS とは定められた時間間隔 (デッドライン) 以内にタスク処理完了可能な動作を行うことができる OS のことである．リアルタイム OS の特徴として，最悪実行時間 (WCET : Worst Case Execution Time) とデッドラインに基づいたスケジューリングが行われることや汎用 OS と比較してより細かい時間で制御できることが挙げられる．

WCET とは最悪の条件下でタスク実行にかかる時間のことである．最悪の条件としては割り込みやキャッシュミスなどの実行時に発生する要因が全て起こるものとすることである．WCET は実行前の静的スケジューリングに必要な情報となるためタスク実行前にあらかじめ計算しておいたものを使用する．

リアルタイム OS の特徴により細かい時間制御が可能となるため WCET がデッドラインに収まるように正確な時間にタスク処理を終えることが可能となる．

2.2 リアルタイム

リアルタイムはその精度の要求によってハードリアルタイム，ファームリアルタイム，ソフトリアルタイムの 3 種類に分類される．本節ではそれぞれの特徴について記述する．

- ハードリアルタイム

  ハードリアルタイムはデッドライン以内に終了しなかった場合，システムが破綻するものを指す．例として，自動車のブレーキが挙げられる．ブレーキをかけて目標の地点で停車できない場合，追突が起こり得る．そのため，ブレーキの処理は必ずデッドライン以内で

完了するように設計しなければならない．このようにデッドライン以内に処理を完了させるシステムをハードリアルタイムと呼ぶ．

- ファームリアルタイム

  ファームリアルタイムはハードリアルタイムと違い，デッドラインオーバーによってシステム自体が破綻することはないが，システムの価値が無くなるものを指す．例として，電車の遅延情報表示システムが挙げられる．電車の事故などで定刻から発車が遅延した場合，遅延情報が表示される．しかし，遅延情報の表示が実際の電車が到着する時刻を過ぎた場合には遅延情報の価値が無くなる．このようにある時刻を過ぎるとシステムとしての価値が無くなるものをファームリアルタイムシステムと呼ぶ．

- ソフトリアルタイム

  ソフトリアルタイムはデッドラインを越えるとシステムの価値が徐々に低下するものを指す．例として，銀行の振り込み処理が挙げられる．振り込みを行う際に振り込み情報を入力してカードや現金を読み込ませる．しかし，デッドラインオーバーによって振り込み情報の入力や確認の応答に時間がかかるとユーザから見て価値が低

下していくということになる．このようにデッドライン以内に処理が完了しない場合でもシステムは破綻しないが価値が低下するものをソフトリアルタイムと呼ぶ．

本研究ではハードリアルタイムで動作するシステムを対象とする．

2.3 ユニプロセッサ用RTOSとマルチプロセッサ用RTOS

2.3.1 ユニプロセッサ

一つのシステムにプロセッサが一つだけあるものをユニプロセッサと呼ぶ．従来，RTOSを動作させる場合にプロセッサを複数搭載するマルチプロセッサでは正確な時間管理の困難性からリアルタイム性の保証が困難であった．しかし，近年ではプロセッサのマルチコア化が進んでいるため性能向上を目的としたRTOSのマルチプロセッサ対応が進んでいる．

2.3.2 マルチプロセッサ

RTOSを動作させるプロセッサ環境について述べる．マルチプロセッサは構成するプロセッサの種類により，図2.2のようにホモジニアスマルチコアとヘテロジニアスマルチコアに分類される．

- ホモジニアスマルチコア

  ホモジニアスマルチコアは同種のコアを複数搭載したマルチコアで

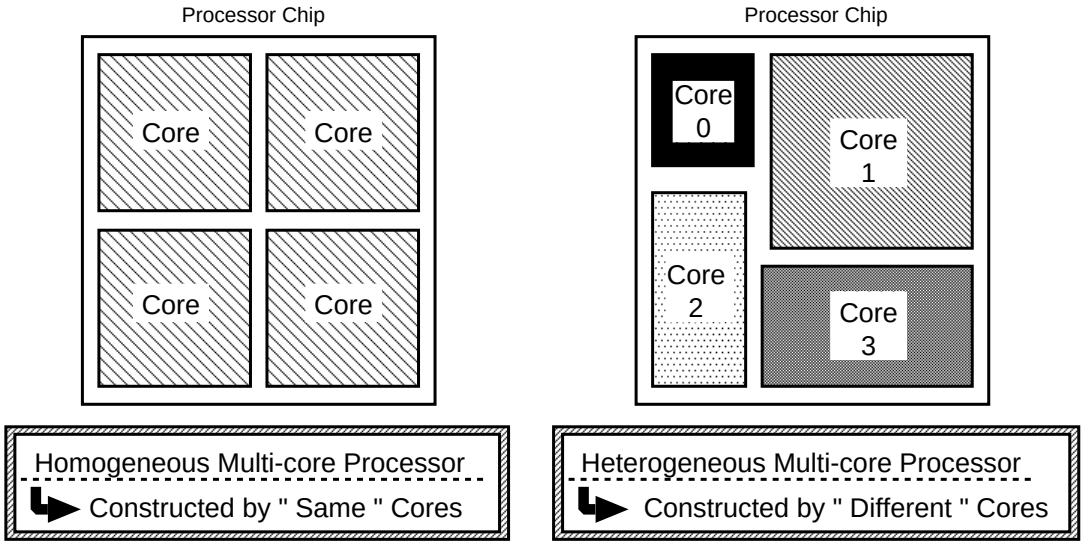

図 2.2: ホモジニアス構成とヘテロジニアスマルチコア構成のプロセッサ

ある．ホモジニアスマルチコアの利点としては，コアや命令セットの違いを意識せずにプログラミングできることや同種のコアを使えるので HMP に比べ設計が容易であることがが挙げられる．欠点としては，同種のコアを用いるためコアに応じた最適な処理を常に行えないことが挙げられる．

- ヘテロジニアスマルチコア

  ヘテロジニアスマルチコアは異種のコアを混載したマルチコアのことである．ヘテロジニアスマルチコアの利点はコア切り替えによりタスクの規模毎に最適な消費電力のコアで処理を行うことができることである．欠点としては CPU やタスクの制御をしなければなら

ないためシステムが複雑化することである．

本研究では消費電力の削減とリアルタイム性の両立が目的である．そのため，タスク毎に最適な消費電力で処理可能なヘテロジニアスマルチコア構成を用いる．詳細は次節で記述する．

2.4 HMP

2.4.1 HMP における命令セット

HMP の実装方法としては一種類の命令セットのみで実装する単一 ISA と複数の命令セットを用いる異種 ISA が存在する．それぞれの特徴について記述する．

- 異種命令 HMP

  異種 ISA の利点は命令セットを複数使用できる単一より高性能にすることが可能となることである．欠点としてはプログラマが命令セットの違いを意識して実装する必要があることである．

- 単一命令 HMP

  単一の利点はコア毎の命令セットの違いがないためプログラムが組みやすいこととコア切り替えが行えることである．欠点としては命令セットの制約によって最大限の性能が出せない場合があることで

ある．

本研究ではコア切り替えによって省電力化を行うが，異種 ISA の場合のコアはあらかじめ想定されたタスクに特化させた処理を行うためコア切り替えは行わない．そのため，本研究では様々な種類のタスクにも対応できる単一 ISA を用いる．

2.4.2 単一 ISAHMP 上での RTOS

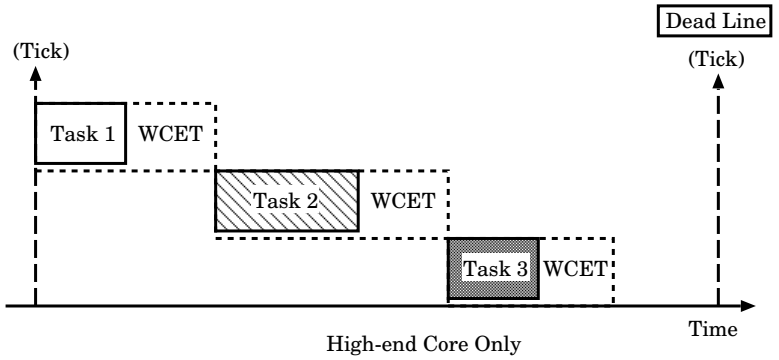

図 2.3: 通常のスケジューリング手法

RTOS を動作させるにあたって，信頼性の評価として応答時間が一定の間隔を守ることを用いる．RTOS においてタスクは一般的に周期実行される．周期実行とは複数のタスクをそれぞれに決められた周期時間で呼び出されタスクを一つづつ繰り返し実行していくことである．周期実行中の全てのタスクが 1 回周期実行されるまでを 1 周期とする．1 周期を

実行するまで許容される時間をデッドラインと呼び，これを用いてタスクスケジューリングが行われる．タスクスケジューリングは各実行タスクのWCETを元にそれらがデッドライン以内に実行が終了するようにタスクを割り当てる．スケジューリングを行った例が図2.3である．しかしこの手法では必ずタスク実行がWCETよりも終了するため，無駄な待ち時間が発生してしまう．それを有効活用するため，本研究では単一ISAHMP構成を効率的に利用することで省電力化を行う．単一ISAHMPにおいては複数のコアを用いるため専用のスケジューリング手法を用いる．この手法については4章で詳しく説明する．

3　関連研究

東京大学の畑中らによってHMPにおいて動的タスクスケジューリングを行う手法[4]が提案されている．この手法は実行時間が入力によって動的変動するタスクを周期実行する場合を想定する．タスクを周期実行する際にタスクの実行開始を入力よりも遅らせ，後続の複数のタスクと連続して実行するまとめ実行を行う．また，消費電力の期待値を求めデッドラインに応じてコア切り替えを行う．この手法によりまとめ実行でプロセッサの実行状態の遷移にかかる消費電力を削減し，消費電力を予測し

ながら実行することで正確な消費電力を見積もってコア切り替えを行うことが可能である．

この手法はコア切り替えのオーバーヘッドを消費電力の計算に組込み，実行時間の一部とすることで対処している．しかしオーバーヘッドはWCET変動の原因となりリアルタイム性の保証に影響が出る恐れがあるため削減する必要がある．本研究室ではオーバーヘッド削減の手法として，コア切り替え機構のハードウェア化を提案する．

4　スケジューリング手法

本研究では，先行研究で提案されたスケジューリング手法の改良を行った．その説明に先立ち，まず一般的なタスクスケジューリングについて述べる．そして先行研究の手法について述べた後，改良の手法について述べる．

4.1　タスクスケジューリング

単一コアで行うWCETを用いた通常のタスクスケジューリングについて述べる．タスクスケジューリングを行う際には，図2.3のようにWCET情報を使用してデッドライン以内にタスクの実行が完了するようにタスクを割り当てる．そして，割り当てたタスクをそれぞれの開始時点から

実行する．しかしこの手法ではタスク処理はWCETよりも終わること起こるため待ち時間が多くなる．先行研究ではこの待ち時間を利用して省電力化する手法を提案した．次節ではその手法について述べる．

4.2 先行研究の手法

先行研究ではHMP構成を用いて図2.3のような待ち時間を利用する手法を提案した．手法は，図4.4のように開始時はローエンドコアで実行し途中からハイエンドコアに切り替えるというものである．

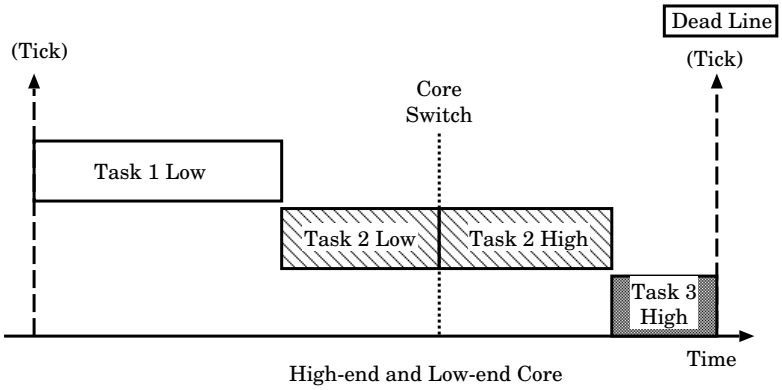

図 4.4: 先行研究のスケジューリング手法

手法を実現するため，先行研究では周期実行するタスクを細分化しそれぞれにチェックポイントを設けた．そしてポイント毎に判定を行うことで動的なコア切り替えを実現した．本節ではその詳細について述べる．

まず，実行タスクを細分化しチェックポイントを設ける．本手法では，

タスクの WCET は事前に計算可能であることを想定している．そこで，本手法ではタスクの WCET 予測を周期単位ではなくより細かい単位で行う．

図 4.5 はタスク細分化の例である．図の例ではタスクに n 個のチェックポイントを設けて周期実行する様子を示している．

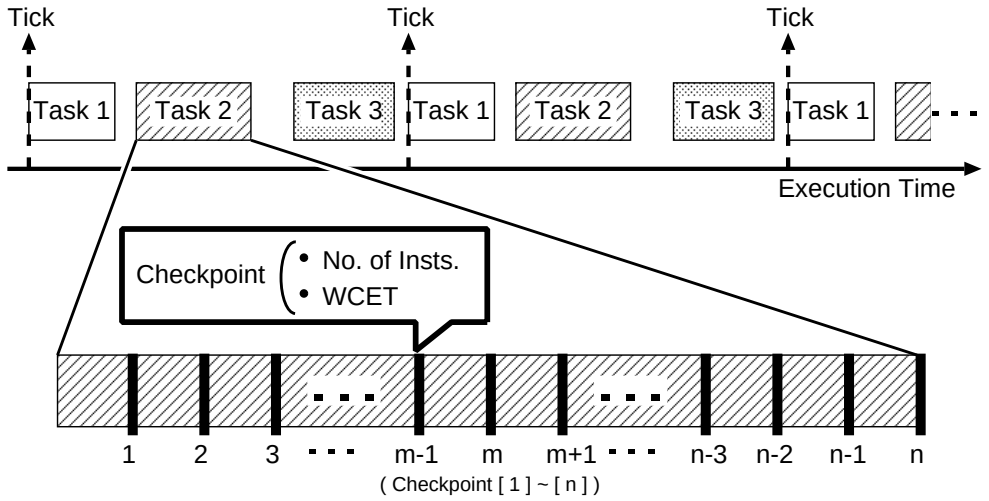

図 4.5: チェックポイント

具体的には，リアルタイムタスクの処理時間単位を解析可能な範囲で細分化し，各チェックポイント時点での実行命令数 (No. of Insts.) と WCET に関する情報を保存する [1]．そして，それらの情報を利用してコア切り替え判定を行う．コア切り替え判定はタスク実行中のチェックポイント毎に行う．コア切り替えのアルゴリズムは以下のようになる．以下のアルゴリズムは先行研究の手法 [1] である．

Dynamic Scheduling Algorithm

```
# 始めに，現行タスクの NoI(実行命令数: number of instructions) が
# Checkpoint[m](m 番目のチェックポイント) に到達しているか否か判定
if ((NoI of Running Task) == (NoI of Checkpoint[m])) {

  # Δ t[m+1]: Checkpoint[m] から Checkpoint[m+1] までの
  #         : ローエンドコアでの最悪実行時間

  Δ t[m+1] = (WCET of Checkpoint[m]) - (WCET of Checkpoint[m+1]);


  # T[m+1]: Checkpoint[m+1] まで実行した際の予測総実行時間

  T[m+1] = (Execution Time of Running Task) + Δ t[m+1];


  # R[m+1]: Checkpoint[m+1] からデッドラインまでの予測残り時間

  R[m+1] = (Deadline of Running Task) - T[m+1];


  # R_high[m+1]: ハイエンドコア (HC) 上で現行タスクを実行した場合の
  #            : Checkpoint[m+1] から処理完了までの最悪実行時間

  R_high[m+1] = (WCET on HC) - (WCET of Checkpoint[m] on HC)

                             + Core Switching Overhead(※ 2) ;
                             # コア切り替えに掛かる時間の考慮

  # コア切り替えの判定

  if (R[m+1] < R_high[m+1]) {
    Flag of Core Switching = 1;
  } else {
    Flag of Core Switching = 0;
  }

  m ＝ m ＋ 1;
}
```

図 4.6: 動的スケジューリングアルゴリズム

この手法はチェックポイントを通過する毎にローエンドコアで実行を続けた場合とハイエンドコアに切り替えて実行した場合の計算を行い，ローエンドコアではデッドライン以内に処理できないと判定した場合のみコア切り替えを行う．

4.3 提案手法

先行研究の手法は一度コア切り替えを行うと次の周期までローエンドコアを使用しないというものであった．しかし，ハイエンドコアに切り替え後に最悪条件が重なることは少ないため、実際にはハイエンドコアで高速に処理することでタスク処理が早く終わることが多いと考えられる．この場合にデッドラインまでの待ち時間が長くなるため，より電力効率の良いローエンドコアで処理すれば待ち時間を有効に利用できる．そのため，本研究の提案手法として，ハイエンドコアでの実行中にもコア切り替え判定を導入する．具体的なアルゴリズムは以下のようになる．

Proposed Dynamic Scheduling Algorithm

```
# 始めに，現行タスクの NoI(実行命令数: number of instructions) が
# Checkpoint[m](m 番目のチェックポイント) に到達しているか否か判定

if ((NoI of Running Task) == (NoI of Checkpoint[m])) {

  # Deadline: タスクに与えられた実行可能時間

  # Δ t[m+1]: Checkpoint[m] から Checkpoint[m+1] までの
  #         : ローエンドコアでの最悪実行時間

  Δ t[m+1] = (WCET of Checkpoint[m]) - (WCET of Checkpoint[m+1]);


  # T[m+1]: Checkpoint[m+1] まで実行した際の予測総実行時間

  T[m+1] = (Execution Time of Running Task) + Δ t[m+1];


  # R[m+1]: Checkpoint[m+1] から処理完了までを
  #       : ハイエンドコア (HC) 上で現行タスクを実行した場合の
  #       : 予測総実行時間

  R[m+1] = (WCET of Checkpoint[m+1] to End on HC) + T[m+1];
                              + Core Switching Overhead ;
                              # コア切り替えに掛かる時間の考慮

  # コア切り替えの判定

  if ((Deadline) < R[m+1]) {
    Flag of Core Switching = 1;
  }
  else if(lock <= 0){
    Flag of Core Switching = 0;
  }


  m ＝ m ＋ 1;
  if(lock > 0)
   lock--;
}
```

図 4.7: 提案アルゴリズム

このアルゴリズムではコア切り替え回数抑制用のカウント変数を用いる．本研究ではこの変数をロック変数と呼称する．コア切り替えを抑制する理由として，コア切り替えには処理にオーバヘッドが発生する．これはハードウェアで実装した場合でも発生する．ロック変数を用意しない場合，コア切り替えが多発し無駄な電力を消費してしまうこととなる．そのためハイエンドコアへ切り替えた後，50 分の 1 周期分のカウントを行い，その間は切り替えを行わないように設定した．この 50 分の 1 という値は 50 分の 1 から 500 分 1 のまで 50 ずつ変化させた値や 10 分の 1，25 分の 1，75 分の 1 の値をそれぞれ比較して最適なものとして実験的に求めたものである．しかしながら，この値は本ベンチマーク環境での実行下においてコア切り替え抑制の効果が得られた場合の値であり，実際のリアルタイムタスクにおいては未検証であるため最適であるとはいえない．カウントする値の最適値の設定は今後の課題となる．

4.4 提案手法のハードウェア化

先行研究ではハードウェアスケジューラを提案した．本研究での提案手法をハードウェア化するにあたり，図 4.8 のようにハードウェアの改良を行った．変更した部分は Dynamic Scheduler Unit の Task info Reg と

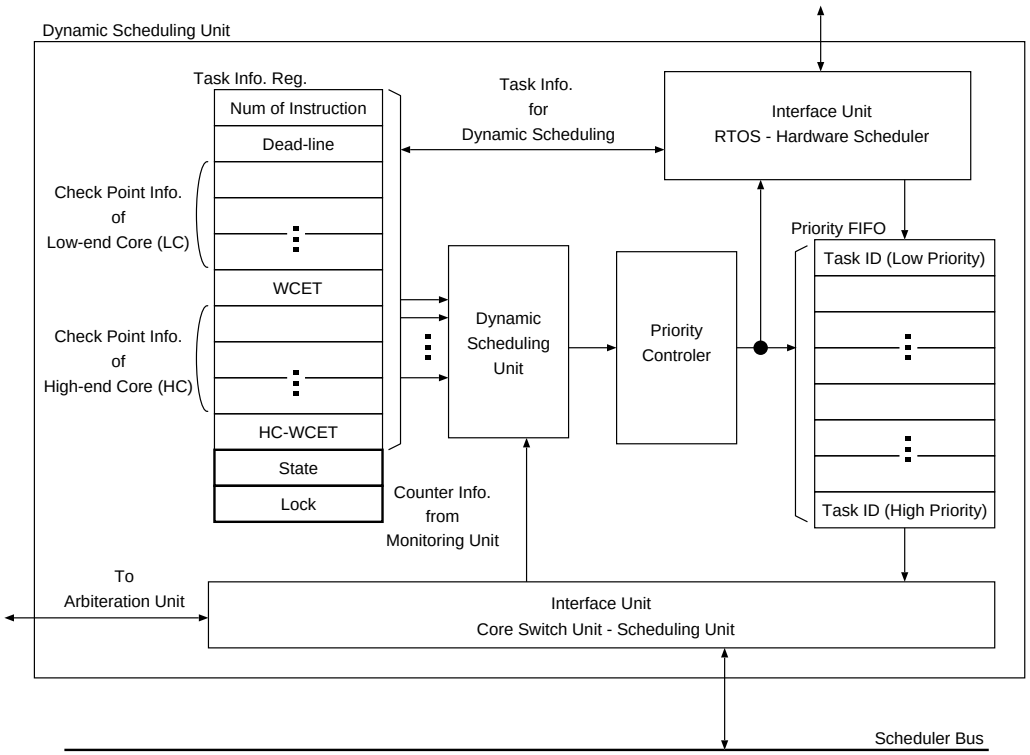

図 4.8: Dynamic Scheduling Unit

Dynamic Scheduler 部である．Task info Reg には実行コアの情報とコア切り替え回数抑制用ロック変数を追加した．Dynamic Scheduler には図 4.9 のように WCET がデッドライン以内に収まっている場合にロック変数が 0 になるまでロックを行う機構を追加した．

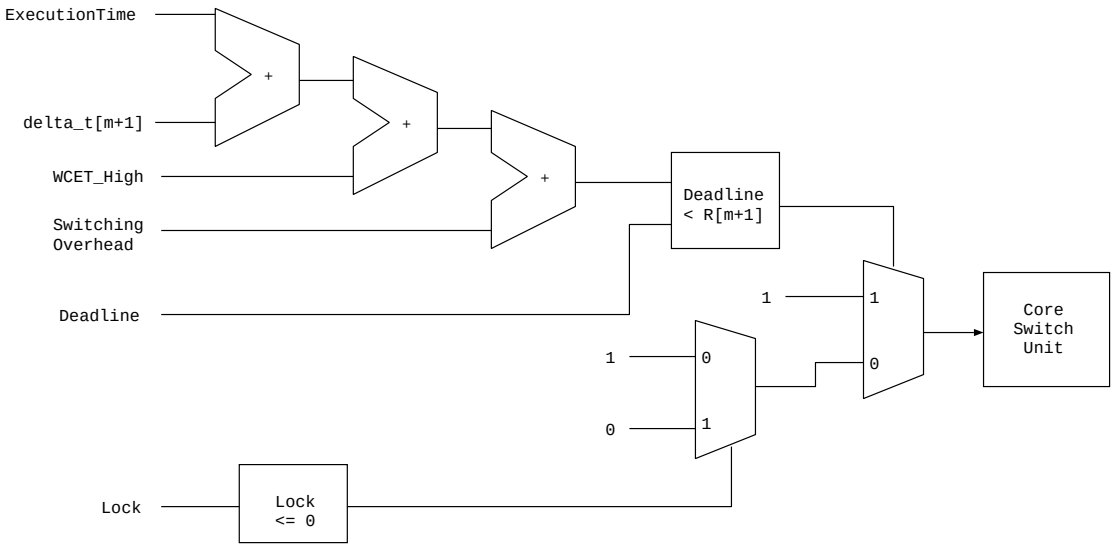

図 4.9: Dynamic Scheduler

5 性能評価

5.1 評価環境

提案手法の性能評価を行う環境について記述する．本研究ではスケジューリング手法の検証のため C 言語によるシミュレーションを行った．リアルタイム処理の検証には時間精度の高い処理機構が必要となるため，シミュレーションで評価を行うことでより正確な評価を行うことができる．

表 5.1 に評価環境の詳細を示す．周波数はハイエンドコアをローエンドコアの 4 倍の周波数とした．これは実験的に固定値で実装したものであるため最適な比ではない．

表 5.1: 評価環境

Clock Frequency	
High-end Core	2.8GHz
Low-end Core	700MHz
Target task	
タスク周期	1ms
チェックポイント数	100

5.2 ベンチマークプログラム

ベンチマークプログラムは周期タスクを想定している．表 5.1 に示したように，タスクの実行周期は 1ms とする．これはリアルタイムシステムの周期実行の評価で用いられた値 [7] である．タスクはハイエンドコアで理想的に動作して周期の 10 分の 1 で終了するものを想定した．この値は周期に対して十分小さい値であるものとした．この値とすることで同一周期内で他のタスクを実行するための処理時間が確保できる．そして，WCET の測定手法が未確立であるため，今回は WCET は理想的な処理時間の 4 倍と想定したもので実装する．先行研究の手法では最初のチェックポイントで必ずデッドラインをオーバーする．そのため，デッドラインはハイエンドコアの WCET を上回り，なおかつ余裕のある値とすることでローエンドコアから実行できるようにした．

なお，提案手法は相互にコア切り替え可能であるため，マルチタスク

で実行することを想定したものとしてデッドラインを WCET と同じ値に設定した場合の評価も行う．

ベンチマークは空ループ処理を実装した．ループ構造とすることで，ループ末尾にチェックポイント判定機構が追加できる．そのためチェックポイント設定の簡単化に繋がる．

ベンチマークはタスク実行のシミュレーションを行う．2.8GHz の場合に 0.1ms で終了する値となる，総実行命令数が 28 万命令となるタスクを実行するものとする．そして 1 ループする毎に 1 サイクルあたりの実行命令数 (IPC : Instruction Per Cycle) を総実行命令数から減算することで 1 ループが 1 サイクルの計算となる．そして残実行命令数が 0 となった場合にタスク実行が終了する．この時にハイエンドコアとローエンドコアで実行したサイクル数から消費電力を求める．

タスク実行の際は IPC を変化させて実行する．このベンチマークはハイエンドコアで理想的に動作して周期の 10 分の 1 で終了するものを想定しているため，タスクは IPC におけるハイエンドコアの理想値で動作した場合に 0.1ms で終了するものとする．表 5.2 の IPC の理想値に負荷をかけ，WCET の値になるまで負荷変動させて測定する．負荷変動は IPC の値を 5%刻みで減らすことで実現する．75%まで減少した場合，理想値

の 4 分の 1 の値となるため WCET で実行する値となる．

コア切り替えに伴うオーバーヘッドはタスク周期の 1000 分の 1 サイクルとする．この値はチェックポイント間隔よりも小さくなる値として設定した．チェックポイント間隔自体が大きい値であるため，ハードウェアスケジューラを仮定していることからチェックポイントより小さい値となることは妥当である．以上のプログラムからスケジューリング手法が動作しているか確認する．

今回の評価では，ハイエンドコアのみの動作，先行研究の手法，ロック機構のない提案手法，ロック機構を付けた提案手法の 4 種類を評価する．

評価の条件を以下の表にまとめる．

表 5.2: 評価条件

タスク周期	1ms
Deadline	0.6ms
実行時間 (理想値))	0.1ms
WCET(High)	0.4ms
WCET(Low)	1.6ms
IPC	
理想値 (HC)	4
理想値 (LC)	1
WCET(HC)	1
WCET(LC)	0.25

5.3 評価結果

ベンチマークプログラムにより，ローエンドコアとハイエンドコアの負荷変動させた場合のスケジューリング手法の効果を確認した．また，負荷変動させることで，各コアの処理能力によらず効果があるかを確かめた．

評価結果を以下の図 5.10，図 5.11，図 5.12 に示す．消費電力は各コアで動作したクロック数にハイエンドコアには 2.8GHz の逆数，ローエンドコアには 700MHz の逆数をそれぞれ乗算することで実行時間を算出し，その値に各コアの平均ワット数を乗算することでワット時として消費電力を算出する．平均ワット数は HMP における消費電力の削減についての研究 [5] より，ハイエンドコアは 46.44W とする．また，消費電力は周波数に比例し電圧の 2 乗に比例することからローエンドコアの消費電力を算出する．周波数を 4 分の 1 にすると電圧は約 54.2%まで下がる [6]．そのため，消費電力はハイエンドコアのワット数に 4 分の 1 と 0.542 の 2 乗を乗算したものとなる．よって，ローエンドコアの値は計算より得られた値である約 3.44W とする．

各図の横軸は IPC にかかる負荷の割合である．IPC の減少率が 75%になった場合を 100%の負荷としてグラフに記載する．0%は負荷が無い状態であるため実行時間は理想値となる．負荷が 100%かかる場合は IPC が

4 分の 1 となるため，実行時間は WCET となる．また，図 5.10，図 5.11，図 5.12 の 30%以下は常にローエンドコアで実行されたため，40%の値のみ記載して他は省略する．

結果より，ローエンドコアが WCET で動作しない場合は先行研究の手法より性能が向上する傾向がみられ，平均で約 7%程度の消費電力削減に成功した．

また，結果により 60%前後の中程度の負荷での効果が高いことがわかった．常に WCET で動作するという状況は起こらないと考えられるため，提案手法の効果は高いと言える．

WCET とデッドラインを同じにした場合の評価は図 5.13，図 5.14，図 5.15 に示す．

評価より，ロック機構を付けた場合に開始時にデッドラインを満たせない場合でも，提案手法はコア切り替えにより省電力化の効果があるということが示された．

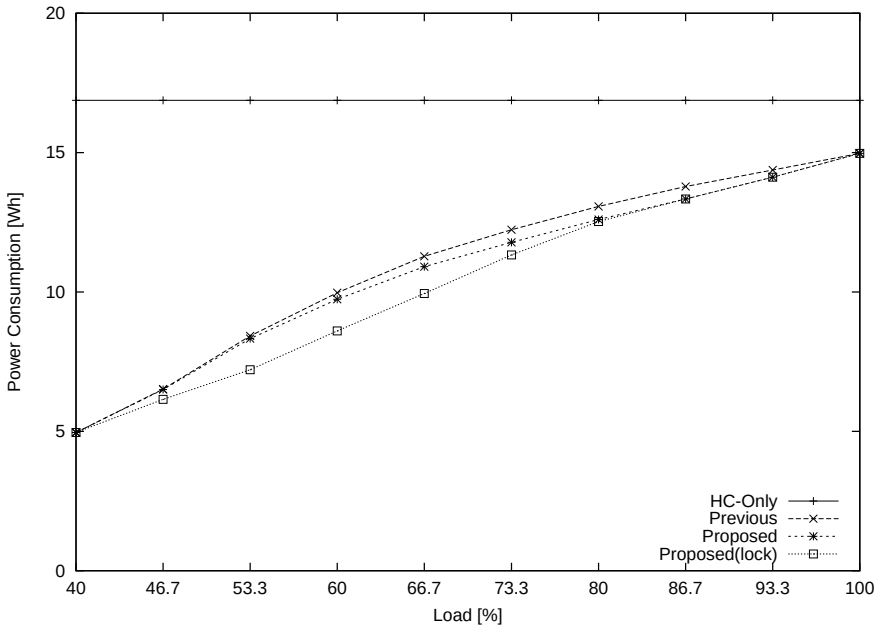

図 5.10: 消費電力

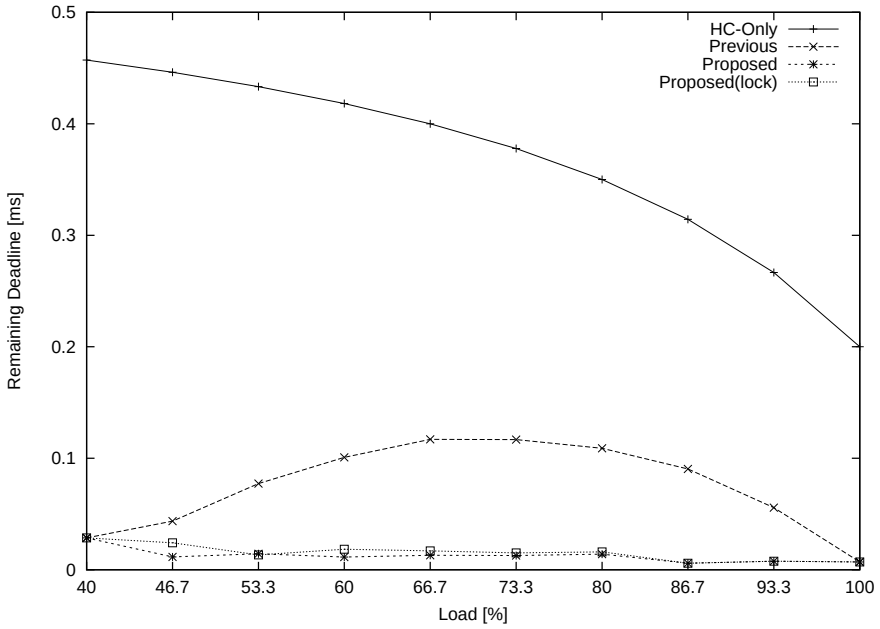

図 5.11: デッドライン残量

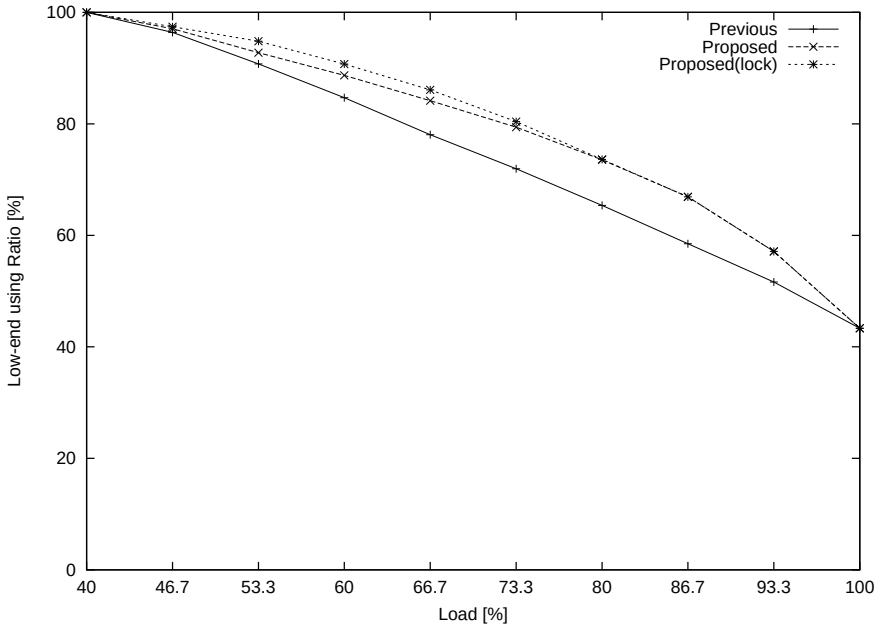

図 5.12: ローエンドコア使用率

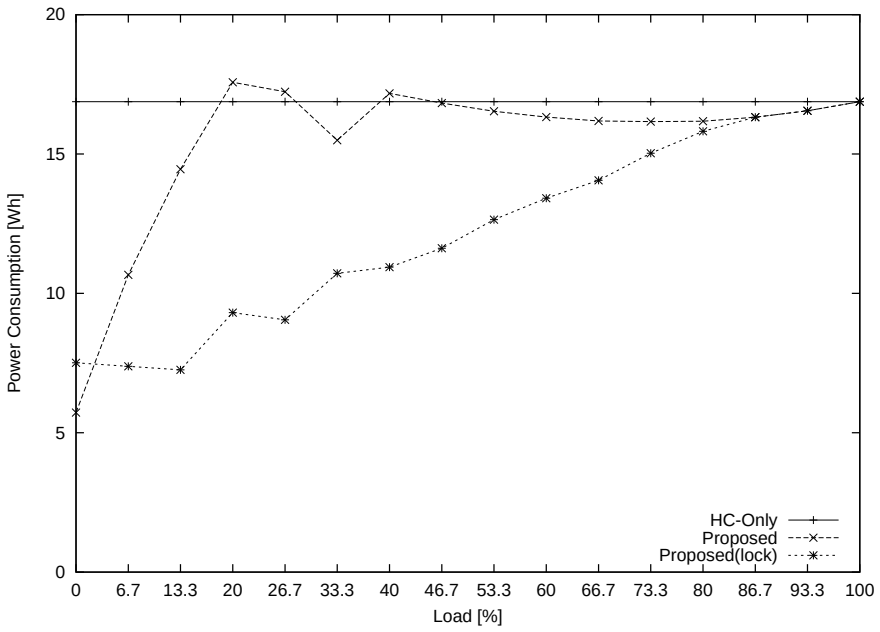

図 5.13: 消費電力 (WCET=Deadline)

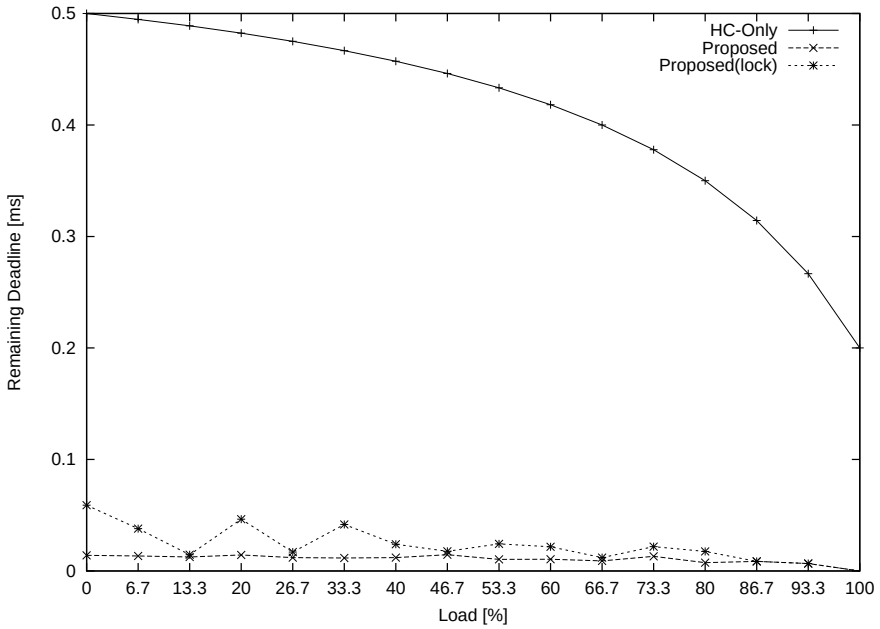

図 5.14: デッドライン残量 (WCET=Deadline)

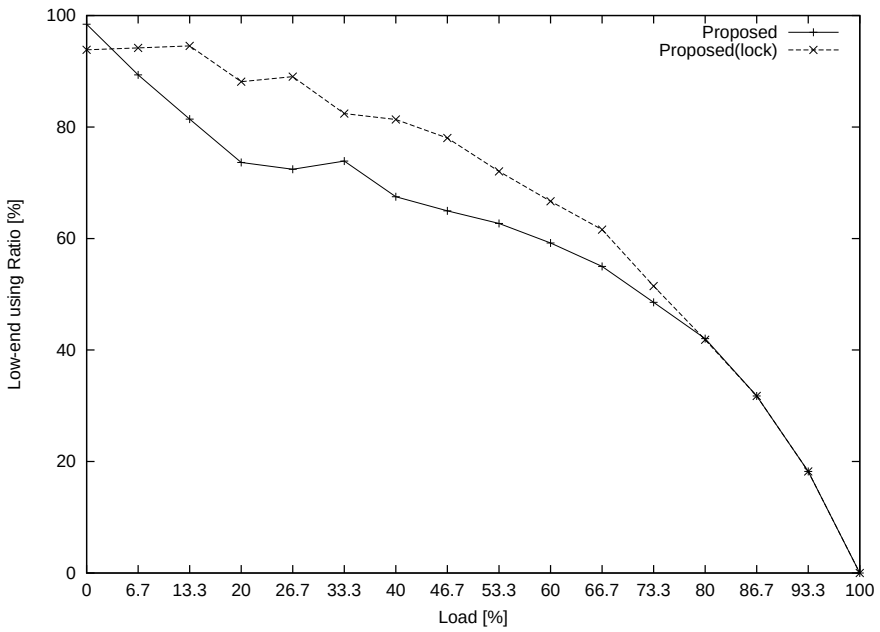

図 5.15: ローエンドコア使用率 (WCET=Deadline)

5.4 考察

図 5.10 より，提案手法は先行研究に比べ 60%前後の中程度の負荷がある場合に省電力性能が高く，負荷が高い場合には効果が薄いということがわかった．これは WCET に近づくにつれて実行時間が遅くなるためハイエンドコアの使用率が上がることによると考えられる．

そして図 5.10 から，ロック機構を設けた場合の消費電力効率が向上していることがわかる．これにより，コア切り替え抑制が機能していると考えられる．

また，図 5.11 と図 5.12 より，先行研究の手法でデッドライン残量が増加している部分でも提案手法ではあまり変化せず，ローエンドコアの使用率が増加していることがわかる．これにより，デッドライン残量のローエンドコアによる有効活用ができたと考えられる．

図 5.13，図 5.14，図 5.15 は先行研究では実行できなかったが，提案手法では消費電力が削減できたため切り替え機構が正しく動作したと考えられる．

6　おわりに

本研究ではハードウェアスケジューラを用いた省電力リアルタイムマルチコアプロセッサのスケジューリングアルゴリズムの改良を行った．この結果，提案手法により先行研究でコア切り替えの問題を解決した．また，手法の改良によって消費電力を平均7%削減できた．

今後の課題として，現状ではWCETを測定するには命令数から概算する方法しかないため，より正確なWCET測定手法を確立する必要がある．また，今回の評価で行ったロック変数と周波数は実験的なものであり，最適な値ではない．そのため今後は最適値を求めることも課題となる．ハードウェア化する際にはキャッシュやメモリなどコア以外の機器を見込んだ実装を行う必要がある．そのため，それらを組込んだ評価を取ることも必要となる．

謝辞

本研究の機会を与えてくださった近藤利夫教授，本研究を進めるあたり多大なご指導を頂いた佐々木敬泰助教，本研究においてご助力して頂いた深澤祐樹研究員に深く感謝いたします．また，本研究を行うにあたって様々なご助力，ご助言して頂いたコンピュータアーキテクチャ研究室の皆様にも心から感謝いたします.

参考文献

[1] 瀬戸 勇介, 中林 智之, 佐々木 敬泰, 近藤 利夫，ハードウェアスケジューラを用いたリアルタイムマルチコアプロセッサの省電力化，第 15 回 組込みシステム技術に関するサマーワークショップ予行集，pp.1-6, 2013.

[2] P. Greenhalgh, Big.LITTLE Processing with Cortex-A15 & Cortex-A7, ARM, http://www.arm.com/files/downloads/big_LITTLE_Final_Final.pdf, 参照 Feb. 23,2015.

[3] B. Jeff, Advances in big.LITTLE Technology for Power and Energy Savings Improving Energy Efficiency in High-Performance Mobile Platforms, ARM, http://www.arm.com/files/pdf/Advances_in_big.LITTLE_Technology_for_Power_and_Energy_Savings.pdf, 参照 Mar. 10,2015.

[4] 畑中 智貴, 中田 尚, 中村 宏：周期実行システムにおける動的省電力タスクスケジューリング, 情報処理学会研究報告, Vol. 2014-SLDM-165, No. 4,pp. 1-6, 2014.

[5] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, D. M. Tullsen. Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction. Proceedings of the 36th International Symposium on Microarchitecture,p.81,December 03-05, 2003.

[6] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, M. L. Scott. Energy-Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scal-

ing. In Proceedings of the 8th International Symposium on High-Performance Computer Architecture,2002

[7] 古川 友樹, 山内 利宏, 谷口 秀夫：組み込みシステム向けの高精度な周期実行制御法の評価, 情報処理学会研究報告, Vol. 2010-SE-168, No. 1,pp. 1-7, 2010.