卒業論文

題目

実行時間の期待値を用いたリアルタイムOSのスケジューリング手法

指導教員

佐々木 敬泰助教

2018年

三重大学 工学部 情報工学科
コンピュータアーキテクチャ研究室

小下　元路(414828)

内容梗概

近年，スマートフォン，自動車の制御システム，ゲーム機本体やコントローラーのようなバッテリ駆動の組み込みシステムが増加しており，同時に高性能化や小型化も進んでいる．バッテリ駆動のシステムにとって，駆動時間の長さはシステムの利便性を大きく左右する要因であり，高性能化によるシステムの消費電力の増加は問題となる．また，システムの高性能化に伴い処理が複雑化し，Worst-Case Execution Time（WCET:最悪実行時間）の増加も進んでいる．特にリアルタイム処理を行う組み込みシステムは，よりデッドライン制約が厳しくなるため，時間的制約を満たすために，高性能なコアが必要となる．コアの性能と電力効率はトレードオフの関係であるため，より高性能なコアを採用することは消費電力増大へとつながる．そこで，リアルタイム性を持つシステムの消費電力を削減するため，ヘテロジニアスマルチコアプロセッサ（Heterogeneous Multi-core Processor:HMP）を利用することで，省電力性能の向上を実現する手法が考案されている．既存の手法はタスク単位でのスケジューリング最適化を行うことで，省電力性能の向上に成功している．また，この手法はコア切り替えをハードウェア実装により行っており，ソフトウェアによるコア切り替えと比較してオーバーヘッドの削減に成功している．しかし，この手法ではタスクの実行順序を考慮しておらず，タスクの実行順序によっては消費電力が増加する場合がある．また，この手法はタスクにチェックポイントを埋め込まなければならないが，コア切り替えの精度がチェックポイント数に依存しており，チェックポイント数を削減すると消費電力が増大する問題がある．加えて，先行研究の手法は実際の実行時間がデッドライン限界付近の場合に，デッドライン制約を満たせなくなるという問題がある．

そこで，本研究では，既存スケジューリング手法にデッドライン制約を常に満たせるように，各タスクの開始直後にもコア切り替え判定を行うように改良を加える．さらに，ハードウェア上で実現可能な，実行時間の期待値を用いたスケジューリング手法を提案し，実行順序を改善することで，実行順序による消費電力の増大を解決し，チェックポイント数を削減した場合に起こる消費電力の増大を抑制する．また，リアルタイム処理を行うカーナビゲーションシステムのシミュレーションを行い，消費電力性能の評価を行う．これにより，消費電力を最大 68%，平均 52%削

減することに成功した.

Abstract

In recent years,battery-driven embedded systems such as smart phones and car control systems, game machine and controllers are increased. At the same time, high performance and miniaturization are advancing. For battery-powered systems, the length of drive time is a factor that greatly affects the convenience of the system. However, power consumption has been increasing due to high performance.

Further, as the performance of the system becomes higher, the processing becomes more complex, Worst-Case Execution Time (WCET) is also increasing. In particular, embedded systems that perform real-time processing satisfy deadline, so a high-performance core is required. The core performance and power efficiency are in a trade-off relationship. So adoption of higher performance cores leads to increased power consumption.

Therefore, in order to reduce the power consumption of a system having real-time nature, a method is propoed by using a heterogeneous multi-core processor (HMP). Existing method succeeds in reducing power consumption by performing task scheduling optimization on a task basis.

However, in this method, power consumption may increase depending on the task execution order. Also, this method must embed checkpoints in tasks. The accuracy of core switching depends on the number of checkpoints. If the number of check points is reduced, average power consumption increases. In addition, the prior method has a problem that the deadline can not be satisfied when the actual execution time is almost same to the WCET.

Therefore, this paper improves scheduling algorithm to always satisfy the deadline constraint. This paper also proposes a scheduling method using expected value of execution time that can be realized on hardware. This solves the problem that the power consumption increases due to the execution order and the problem that the power consumption increases as the number of checkpoints is reduced.

In addition, we simulate the car navigation system that carries out real-time processing and evaluate the power saving performance.

According to our simulation results, the proposed method reduces power consumption by 68% in maximum and 52% in average.

目 次

図 目 次

表 目 次

1　はじめに

近年，スマートフォンや自動車の制御システム，ゲーム機本体及びコントローラーのような，小型かつ高性能なバッテリ駆動の組み込みシステムが増加している．このようなシステムにとって，バッテリーの持続時間はシステムの利便性を大きく左右する要因であり，より長く連続使用が可能であればそれはシステムにとって大きな強みとなる．しかし，高性能化によるシステムの消費電力の増加が進んでおり，高性能と省電力の両立が求められている．また，システムの高性能化に伴いタスクの処理内容が複雑化し，Worst-Case Execution Time（WCET:最悪実行時間）の増加も進んでいる．特にデッドラインまでに処理を完了させなければならないリアルタイムシステムでは，この WCET の増加に伴いデッドライン制約を満たすために，より高性能なコアの搭載が必要となる．コアの性能と電力効率はトレードオフの関係であるため，より高性能なコアを搭載すると，同じタスクを処理する場合でも消費電力が増加する．そのため，高性能と省電力を両立したプロセッサへの要求が高まっている．

そこで，高性能と低消費電力を両立する手法としてヘテロジニアスマルチコアプロセッサ（Heterogeneous Multi-core Processor:HMP）が注目されている．HMP は異なる構成のコアを複数持つプロセッサのことであ

り，各コアの電力対性能が異なる．プロセッサの性能と消費電力の間にはトレードオフの関係があり，例として処理速度を 4 倍にするには 15 倍以上の電力が必要となる．HMP ではタスクの負荷に応じて，処理を行うコアを切り替えることで，高性能と低消費電力の両立が可能となる．そのため，デッドライン制約を満たすために性能面で妥協のできないリアルタイム処理において，消費電力の削減を行うための手法として HMP が注目されている．このような構成は ARM の big.LITTLE[3] で商用として実際に採用されており，省電力化に成功しているという評価結果 [4] もあり，有用性が高い．しかし，ソフトウェア実装によるコア切り替えでは，タスクのコンテキストスイッチによるオーバー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，デッドラインまでの時間を有効に使い切れず，リアルタイム処理においては消費電力の面で最適化の余地がある．そこで，先行研究 [1][2] では，スケジューリング機構のハードウェア化が行われている．しかし，先行研究のスケジューリング手法は単一タスク内で完結しており，タスクの実行順序へのアプローチは無いため，実行順序によっては大きく消費電力が増加する．また，先行研究にはデッドライン制約を満たせない場合が存在する．

そこで，本研究では，そして，複数タスクを周期実行するリアルタイム

システムにおいて，タスクの実行時間の期待値と WCET の差に着目し，タスクの実行順序を最適化することによる消費電力削減手法を提案する．また，先行研究の手法がデッドライン制約を満たせない問題を解決するため，タスク開始直後にもコア切り替え判定を行う手法を提案する．また，先行研究で評価に用いていたシミュレータは，特定命令数の単一タスク処理において検証を行っている．しかし，複数タスクをデッドラインまでに実行する場合や，タスクへの入力による実行時間の変動特性が考慮されていない．そこで，本研究の評価では，先行研究とは異なる，新たに作成したシミュレーションプログラムを用いる．本研究では，新たに作成したシミュレーションプログラムを用いて先行研究のスケジューリングアルゴリズムを評価し，さらなる改善案を提案する．また，本研究では，big.LITTLE と同様にハイエンドコア (HC) とローエンドコア (LC) の 2 つのコアを切り替えて処理を行う HMP を想定する．以下，第 2 章で HMP とリアルタイム処理に関する概要を述べ，第 3 章，第 4 章で既存の手法とその問題点について述べる．第 5 章より本研究で提案する手法について述べ，第 6 章で提案する手法の評価方法とその結果について述べる．最後に，第 7 章でまとめと今後の展望について述べる．

2　マルチプロセッサ用リアルタイムOS

既存の手法を説明するに先立って，リアルタイム処理とマルチコアプロセッサの概要について述べる．次に，HMP 環境におけるリアルタイム処理について述べる．

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（デッドライン）までにタスクの処理が完了するように処理を行うOSのことである．本研究ではタスクの実行可能時点からデッドラインまでの時間を周期と呼び，周期毎に特定のタスクを実行する周期実行について扱う．リアルタイムOSの特徴として，実行するタスク全てがWCETで実行されることを想定してスケジューリングを行うことが挙げられる．WCETとは，あらゆる条件が最悪の場合にタスク実行にかかる時間のことである．すなわち，割り込みやキャッシュミス等，全てにおいて悲観的な場合を想定した場合の実行時間である．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値を使用する．

2.2　リアルタイム性

リアルタイム処理は，その精度の要求により，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，各リアルタイム性の特徴について説明する．また，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．

2.2.1　ハードリアルタイム

ハードリアルタイムとは，デッドライン制約を満たせ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．ブレーキの作動が想定よりも遅れ，目標の停止距離で停止できない場合，追突が起こりえる．そのため，ブレーキの動作処理は必ずデッドライン以内で行われなければ，ブレーキというシステムが破綻する．このように，ハードリアルタイムでは，デッドライン制約を満たせなかった場合，生命の損失や機械の故障のように，重大な損害が起こりシステムが破綻するものを指す．

2.2.2　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異なり，デッドライン制約を満たせなかった場合にシステム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，動画の再生が挙げられる．圧縮された動画を再生する際には復号化を行わなければならないが，復号化を行なっているフレームを表示するべき時間が過ぎてしまうと，そのフレームの復号化処理の価値は失われてしまう．このようにファームリアルタイムでは，デッドライン制約を満たせなかった場合，システムの価値が無くなってしまうものを指す．

2.2.3　ソフトリアルタイム

ソフトリアルタイムは，デッドライン制約を満たせなかった場合にシステムの価値が徐々に低下するものを指す．ソフトリアルタイムにおいてデッドラインとは，システムの価値が下がり始めるタイミングを示している．例としては，ウェブブラウザソフトウェアが挙げられる．ウェブブラウザソフトウェアはデッドライン制約を満たせなかった場合にもシステムが破綻したり，価値そのものがなくなってしまうことはないが，応答時間が長いほどユーザーを待たせてしまうことになり，徐々に価値が低下する．このように，デッドライン制約を満たせなかった場合，徐々にシステムの価値が低下していくものをソフトリアルタイムと呼ぶ．

2.3 マルチコアプロセッサと RTOS

プロセッサに 1 つのコアを搭載したものをユニプロセッサ，複数のコアを搭載したプロセッサをマルチコアプロセッサと呼ぶ.

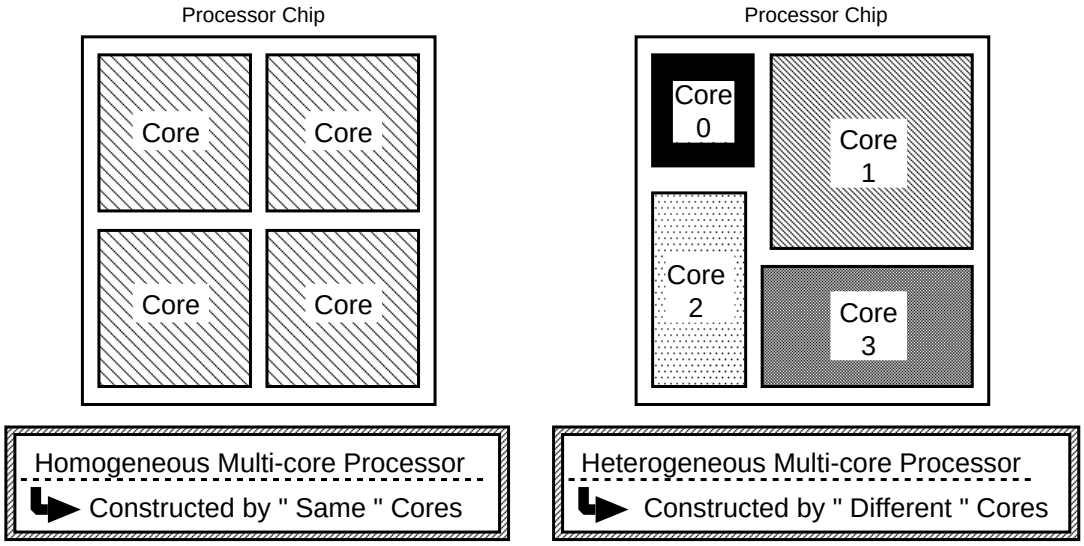

図 2.1: マルチコアプロセッサ

マルチコアプロセッサは構成するプロセッサの種類により，図 2.1 のようにホモジニアスマルチコアプロセッサと HMP に分類される．ユニプロセッサ上でリアルタイム処理を行う場合，全タスクが WCET で実行された場合にもデッドライン制約を満たせる性能のコアを持つユニプロセッサが必須となる．しかし，タスク処理は実際には WCET よりも早く終わること多いため，待ち時間が発生する．そのため，この待ち時間を利用

して省電力化を行う手法として，HMP が注目されている．本節では，通常のマルチコアプロセッサと HMP について述べる．

- ホモジニアスマルチコアプロセッサ

  ホモジニアスマルチコアプロセッサは同種のコアを複数搭載したマルチコアプロセッサである．利点として，1 種のコアにより構成されるため，開発コストがヘテロジニアスマルチコアプロセッサに比べて低いことが挙げられる．しかし，1 種のコアしか持たないため，タスクによってコア性能の過不足が出ることが挙げられる．そのため，省電力性能の面で後述する HMP が注目されている．

- ヘテロジニアスマルチコアプロセッサ（HMP）

  HMP とは，異種のコアを混載したマルチコアプロセッサのことである．HMP の利点として，コアを複数種搭載しているため，タスクを処理するコアを切り替えることにより，タスクの規模毎に最適な性能のコアで処理が可能なことが挙げられる．そのため，組み込み分野において，高性能と低消費電力を両立する手法として注目されている．

本研究では，リアルタイム性を維持しながら消費電力を削減すること

が目的である．そのため，状況に応じて最適なコアにより処理が可能なHMP 構成を用いる．

2.4 HMP と RTOS

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，決められた周期時間毎に 1 つまたは複数のタスクを 1 つずつ実行することである．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 周期の実行時間として許容される時間をデッドラインと呼び，1 周期の実行時間がデッドライン未満となるようにタスクスケジューリングが行われる．組み込みシステムでは，あらかじめ扱うタスクがわかっており，各タスクの WCET を元に，デッドライン以内に実行が終了するようにスケジューリングを行う．また，リアルタイム処理を行うプロセッサは，全タスクが WCET で実行された場合においても，デッドライン制約を満たせる性能が要求される．しかし，タスクが WCET で実行されることは少なく，無駄な待ち時間が発生する．そこで，HMP を用いることで，高性能と省電力を両立させる手法 [1][2] が提案されている．しかし，HMP 上でリアルタイム性を保証することは，タスクのコア切り替え判定やコア切り替え時間により，それらを考慮する必要の無いユニプロセッ

サ上での動作と比較して困難である．そのため，タスクのコア切り替え判定や，コア切り替えによるオーバーヘッドを考慮した手法が先行研究 [1][2] で提案されている．先行研究 [1][2] では，WCET よりも早く実行が終わることにより発生した待ち時間を利用し，省電力性能に優れたローエンドコアで処理を行うことで，省電力化を実現する手法を提案している．しかし，先行研究の手法では，タスクの実行順序によっては無駄な待ち時間が多く発生することがある．そこで，本研究ではタスクの実行順序を考慮したスケジューリング手法を提案する．この手法については第 5 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 を用いたスケジューリングを行う手法 [5] が提案されている．この手法は，入力データに依存して実行時間が変化するようなタスクを，周期的に実行する場合を対象とする．デッドラインが長いことを利用して，タスクの実行開始を入力到着から遅らせて実行し，後続の複数のタスクと連続して行うまとめ実行と呼ばれる手法を用いる．また，タスクの変動確率を用いて消費エネルギー期待値を評価し，稼働コアを動的に変化させる．この 2 つの手法により，消費電力の削減を行っている．しかし，この手法は全タスクの WCET の和とデッドラインの差が大きな，デッドライン制約に余裕のある場合を想定した手法であるため，デッドライン制約の厳しい状況も想定する本研究とは主旨が異なる．また，ソフトウェアによるコア切り替えが実装されている．しかし，ソフトウェアによるコア切り替えにはハードウェア実装されたコア切り替えよりも大きなオーバーヘッドが発生する．

4　先行研究

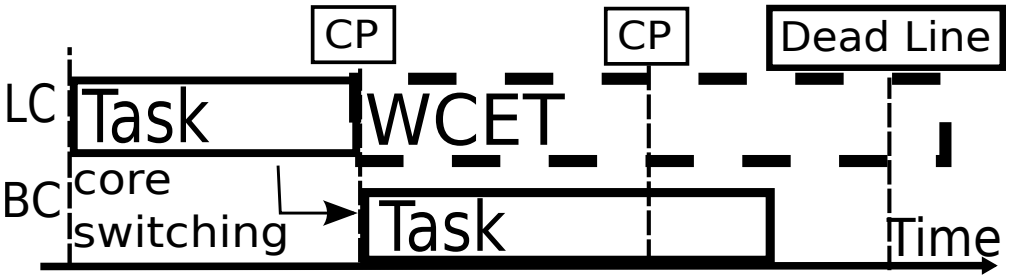

図 4.2: 先行研究のスケジューリング手法

先行研究 [1][2] では，HMP 構成を用いて図 4.2 のように待ち時間を利用する手法が提案されている．先行研究の手法は，開始時はローエンドコアで実行する．そして，途中から状況に応じてハイエンドコアやローエンドコアに切り替えると言うものである．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る．そしてチェックポイントに到達するたびに判定を行うことで動的なコア切り替えを行う．図 4.3 はタスク細分化の例であり，タスクに n 個のチェックポイントを設けている様子を示している．

各チェックポイントには，チェックポイント時点での実行命令数と WCET に関する情報を保存する．コア切り替え時には，このチェックポイントの

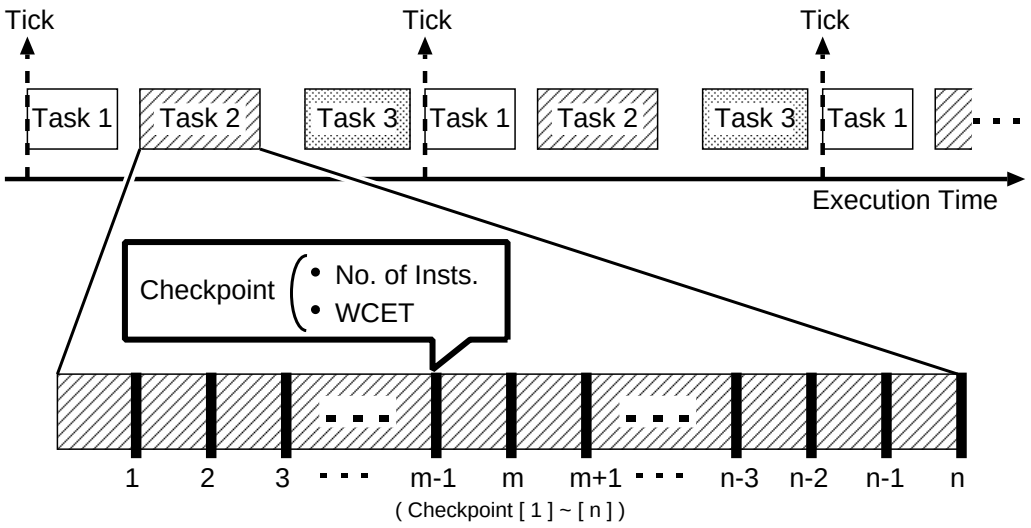

図 4.3: チェックポイント

情報を利用する．タスク実行時には，ローエンドコアを用いて処理を開始し，タスクの実行状態を監視する．処理がチェックポイントに到達した時点でコアを切り替えるかを判断する．瀬戸による先行研究 [1] では，コア切り替えの判断は，ローエンドコアで実行を続けた場合に，デッドライン制約を満たせない場合にはハイエンドコアに変更するというものである．また，嶋谷による先行研究 [2] では，コア切り替えの判断は，現時点でローエンドコアで実行を続ける，もしくはローエンドコアに切り替えてもデッドライン制約を満たせるならローエンドで実行し，満たせないならハイエンドコアで実行をするというものである．これにより，デッドライン制約を満たせる限りはローエンドコアで処理を行うことができ，

大幅に消費電力を削減できる．また，この手法はスケジューリング機構がハードウェア化されているため，ソフトウェアによる切り替えと比べてコア切替時のオーバーヘッドが少ないという特徴がある．

4.1 問題点

先行研究では，タスクの実行時に待ち時間を有効利用することで，省電力化を実現している．しかし，この手法には3つの問題が存在する．1つ目は，複数のタスクを処理する際に，その実行順序によっては，デッドラインまでに余分な待ち時間が多く発生する場合があることである．すなわち，タスクの実行順序によっては大きく消費電力が増加することである．2つ目は，チェックポイントの数とスケジューリングの精度にトレードオフの関係があり，実際にチェックポイントの数を減らすと大きく精度が低下することである．3つ目は，全タスクのWCETの和がデッドラインに近い場合に，デッドライン制約を満たせない可能性があることである．

4.1.1 実行順序による消費電力の増加

本項では，実行順序により消費電力が増加する問題について述べる．

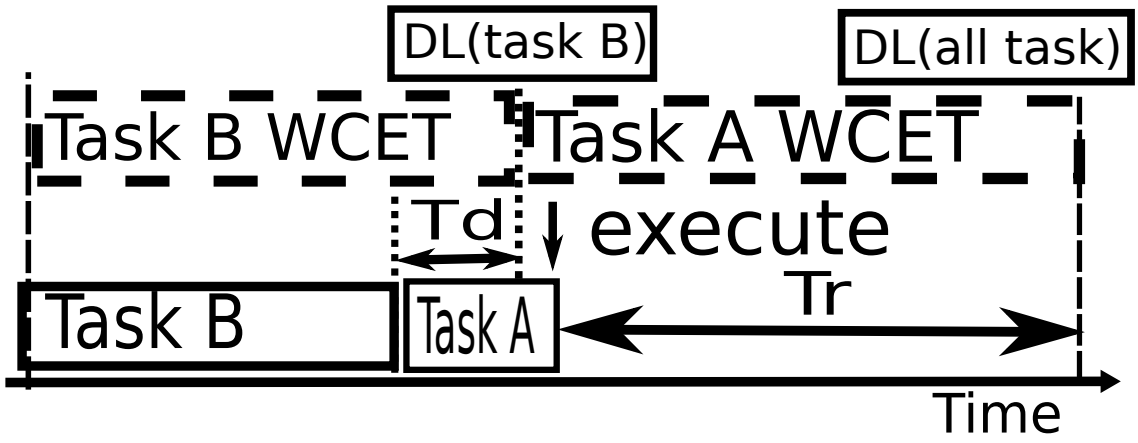

図 4.4: Tr が大きく発生する場合

例として，図 4.4 のように実行順序で 2 つのタスクを実行する場合を考える．図のタスク A は通常は軽い処理を行うが，ある条件を満たした場合に重い処理を行うタスク，タスク B は常に安定した処理量のタスクを想定している．また，点線の四角は WCET を，実線の四角は実際の処理時間を表している．ここで，実際の処理時間と WCET との差の時間を徒労時間 (Time difference:Td)，周期中で最後のタスクの終了とデッドラインまでの時間の差を余剰時間 (Time rest:Tr) と呼ぶこととする．また，図中ではデッドラインを DL と表記する．

リアルタイム処理はデッドライン制約を満たすため，未処理のタスクが全て WCET で処理されると想定した，極めて悲観的なスケジューリングを行わなければならない．そのため，後続のタスクが WCET で実行できる時間を残して，先行するタスクの処理を完了させる．しかし，実際

の実行時間がWCETとなることは少ないため，ほとんどの場合Trが発生する．特に，大きなTdが発生しやすいタスクがある場合，タスクの実行順序によっては，大きなTrが発生することがある．例えば図4.4では，実行前のタスクBの処理は，タスクAの稀にしか発生しない処理によるWCETでの実行のための時間を残して，処理を終了しなければならず，タスクBにとっての事実上のデッドラインが発生する．しかし，実際にはほとんどの場合はタスクAはすぐに処理を終えてしまう．そのため，図4.4の実行後のように，たとえタスクAを全てローエンドで実行しても，タスクAのために用意された時間を使い切ることができない．そのため，大きな余剰時間が発生してしまい，省電力性能が悪化する．前述のようにコアの性能と電力効率はトレードオフであり，消費電力を減らすためにはなるべくローエンドコアでの処理を行うことが望ましい．そのため，Trを極力発生させないようにし，よりローエンドコアで処理を行うことが消費電力の削減につながる．

4.1.2 省電力性能のチェックポイントへの依存

先行研究のスケジューリング手法では，コア切り替え判定タイミングとして，タスク中に一定間隔毎にチェックポイントを設ける必要がある．しかし，頻繁にチェックポイントを設けると，コア切り替えの判定処理が

多く発生するため，タスク処理量が増加する．最小規模のハードウェアで動作させることの多い組み込み分野では，メモリ容量の圧迫も無視できない．反対に，チェックポイント数を減少させるとコア切り替えの精度が低下する．前述したように，リアルタイム処理はデッドライン制約を満たすため，悲観的なスケジューリングを行う．そのため，コア切り替え精度が低下すると，ハイエンドコアでより多く処理するようになり，消費電力が増大する．そのため，チェックポイント数を削減した場合において，それに伴う消費電力の増大を抑制できるような手法が求められる．

4.1.3 デッドライン制約を満たせない場合の存在

先行手法はローエンドコアで処理を開始する．しかし，WCET がデッドライン制約に対して余裕がない場合，ハイエンドコアで処理を開始しなければデッドライン制約を守れないことがある．ローエンドコアで処理を開始する先行手法では，このような場合にデッドライン制約を満たせないことがある．このようなケースは，全タスクの WCET の和とデッドラインが近い場合において，全タスクが WCET 付近の処理時間を要した場合に発生する．特にハードリアルタイム処理においては，このようなデッドライン制約を満たせない可能性があることは致命的な結果につながるため，解決が必須である．

5　提案手法

前章で述べた実行順序により消費電力が増大する問題は，大きなTdが発生した後，後続のタスクがそのTdをローエンドコアでの処理に充てられない，もしくはローエンドコアでの処理に充てたとしても大きなTrが発生する場合に発生する．そこで，本研究では，Tdの期待値を用いてタスクの実行順序を変更する手法を提案する．周期の前半にTdの発生が多い場合は，後続のタスクでTdの有効利用がしやすいと予測できるため，Tdの期待値が大きいタスクから順に実行するようにすることで消費電力の削減を実現する．

5.1　実行時間の期待値を用いたスケジューリング

本節ではまず提案手法について述べた後，その実現方法について述べる．リアルタイム処理はWCETでの実行を前提としたスケジューリングを行う．つまり，Tdの小さなタスクはスケジューラの用意する実行時間と実際の実行時間の差は小さい．Tdの大きなタスクを周期の後半に実行すると，スケジューラは実際の実行時間に対して過剰な時間を用意するため，図4.4のように大きなTrの発生につながる．そこで，Tdの期待値が大きいタスクから実行する手法を提案する．提案手法では，図5.5のよ

うに，Td の期待値が大きいタスクを周期の前半に実行し，Td の期待値の小さなタスクを周期の後半に実行するようにする．これにより，図 5.5 実行後のように，周期の前半に発生した Td を活用して，後半の Td の小さなタスクをローエンドコアでより長く実行することが可能となる．また，周期の後半に大きな Td が発生しにくくなり，図 4.4 のように後続のタスクが Td を利用しきれない場合を削減する．この結果，Tr を削減することができ，より長くローエンドコアで処理を行えるようになるため，消費電力の削減が可能となる．

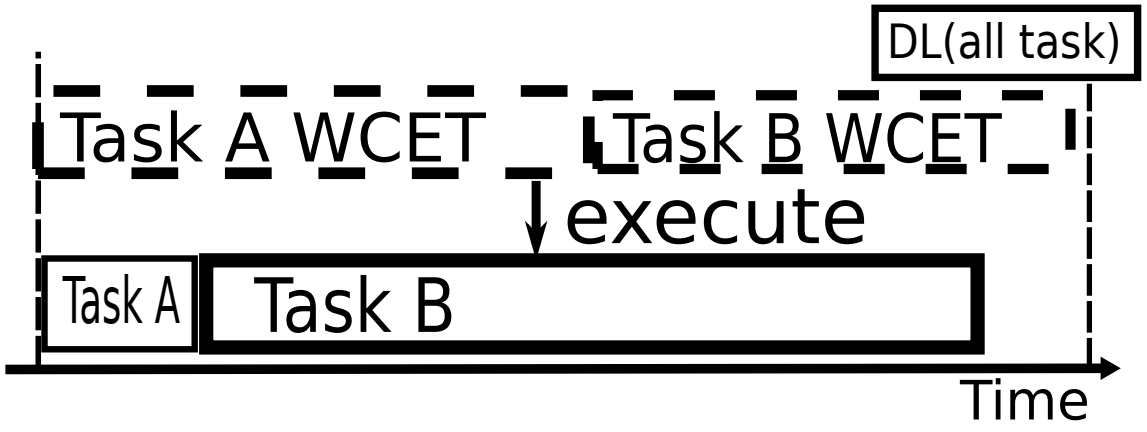

図 5.5: 提案手法により Tr を削減した場合

また，第 4 章で述べたように，先行研究のスケジューリング手法では，コア切り替え判定タイミングとして，タスク中に一定間隔毎にチェックポイ

ントを設ける必要がある．しかし，チェックポイントを減らすとコア切り替え精度が低下し，Td が大きく増加することで，消費電力が大きく増大する．そこで，本手法を用いることで，前述したように大きな Td をなるべく周期前半に発生させ，後半の Td がより少ないと予測されるタスクの処理に充てることができる．そのため，チェックポイント数を削減した場合のスケジューリング精度低下に伴う消費電力増大の影響を抑制できる．

次に本手法の実現方法について述べる．一般に組み込みシステムでは，処理対象のタスクは既知である．特に，信頼性の高いリアルタイム処理ソフトウェアの開発時には，網羅性の高いテストケースや，WCET を解析するためのテストケースでのテストが想定される．また，近年はリアルタイム処理を行うタスクの大規模化，複雑化に伴い，WCET の解析や算出は困難になっている．そこで，本番環境を想定した入力でのテストを十分に行い，その処理時間の最大値に安全係数をかけて最大処理時間とみなす方法 [8] が採用されている．本手法では，このテスト時の情報を用いて，Td の期待値をあらかじめ求める．

事前実行において，タスク i の実行時間を T_i，その実行時間での実行の発生確率を P_i とすると，そのタスクの実行時間の期待値（$E[T_i]$）は $E[T_i] = \sum_{j=1}^{n} T_{i,j} P_{i,j}$ と求められる．ここで，タスク i の最悪実行時間を W_i とす

ると，Td の期待値 $E[Td_i]$ は，$E[Td_i] = W_i - E[T_i]$ となる．ここで，優先度の同じタスクの集合を $T = T_0, T_1, ...$ とすると，提案手法では $E[Td_i]$ の大きいものから実行することで，提案手法を実現できる．

5.2 提案手法のハードウェア化

先行研究では，スケジューリング機構をハードウェア化することにより，コア切り替えのオーバーヘッドを削減している．そのため，提案手法もハードウェア上での実現が可能でなければならない．本手法は，各タスクの優先度の下位ビットに，そのタスクの実行時間の期待値が全タスク中何番目に高いかを格納することでハードウェアによる実現が可能である．先行研究で提案されているハードウェアのブロック図を図 5.6 に示す．図 5.6 の Task Info Reg はスケジューリングに必要な情報を格納するレジスタである．また，このプロセッサは，図 5.6 の Priority FIFO に格納された優先度を守りスケジューリングを行う．本手法を実現するには，図 5.6 の Task Info Reg にタスクの実行時間の期待値の順位を格納し，優先度の下位ビットを期待値の順位の格納に使えばハードウェアに対する改変は不必要である．また，タスクの優先度を保持するビット数を減らさない場合にも，周期内のタスク数を N とした時に，Priority FIFO を高々

$\log 2N$ ビット拡張を行うことで本手法は実現できる．これにより，タスクの実行優先度を守りつつ，提案手法を実現できる．

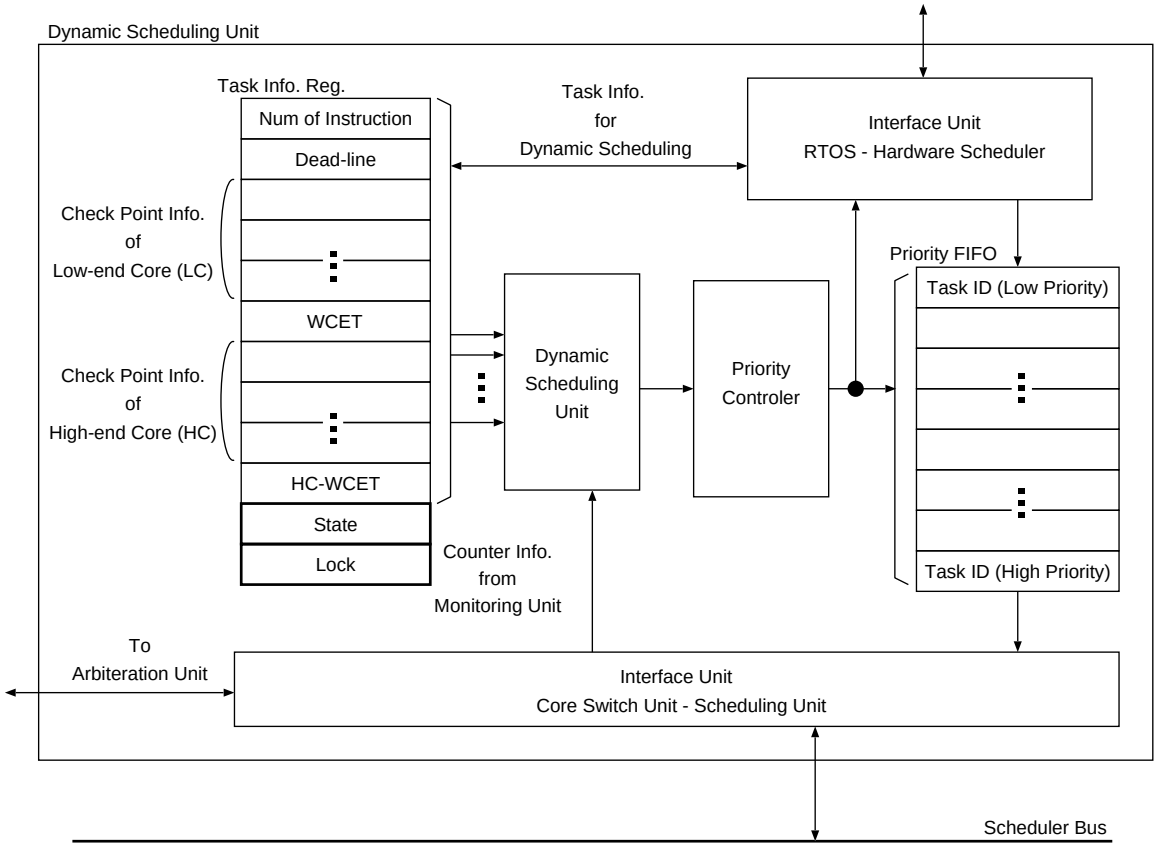

図 5.6: スケジューリング機構

5.3 Start Check Point の設置

先行研究の手法には，ハイエンドコアで処理を開始しなければデッドライン制約を満たせない場合にも，ローエンドコアで処理を開始するた

め，デッドライン制約を満たせないことがあるという問題点がある．しかし，常にハイエンドコアで実行を開始した場合，ローエンドコアで処理を開始してもデッドライン制約が満たせる場合において，次のチェックポイントまで電力効率の悪いハイエンドコアで処理を行うことになり，消費電力の増大につながる．そこで，タスク開始直後にもチェックポイントを設置（start check point:SCP）し，タスク開始直後にもコア切り替え判定を行うことで，必要ならばハイエンドコアで処理を開始できるように改良を行う．

6　性能評価

6.1　評価方法

本研究では，リアルタイム処理を想定したシミュレーションを行い，SCP を適用した島谷による先行研究 [2] の手法と，それに提案手法を適用した場合の消費電力について評価する．シミュレーション対象としてカーナビシステムの周期タスクを想定し，3 つのタスクを実行する．1 つ目は画面表示処理であり，実行時間は安定している．2 つ目は位置情報更新であり，位置情報が急激に変化した場合等，若干実行時間にゆらぎがある．3 つ目は現在地から目的地までのルート計算処理であり，普段は軽い処理であるが，予想ルートから車が離れた場合には再計算が必要となり，非常に負荷のかかる処理となる．以上のタスクについて，次の 3 つの場面を想定して命令数を確率で変動させ，1 万回の試行を行い消費電力の平均で評価を行う．1 つ目の場面は高速道路の走行で，トンネルによる位置情報取得への影響が強く，ルートを間違えることは無い．2 つ目の場面は市街地の走行で，ある程度の速度とルートの複雑性から，若干位置情報取得への影響があり，また，運転者が稀にルートを間違える．3 つ目の場面は住宅街の走行で，位置情報は安定して取得でき，運転者がよくルートを間違える．

設定した命令数及び発生確率を表 6.1，6.2，6.3 に示す．各タスクの WCET の際の命令数は画面表示処理を 620k，位置情報更新を 100k，ルート再計算を 680k とする．

タスク名	命令数（発生確率）
画面描写	620k(90%) , 610k(5%) , 600k(5%)
位置情報	10k(91%) , 50k(6%) , 100k(3%)
再計算	5k(100%)

表 6.1: 高速道路の走行を想定した設定

タスク名	命令数（発生確率）
画面描写	620k(90%) , 610k(5%) , 600k(5%)
位置情報	10k(96%) , 25k(4%)
再計算	5k(99%) , 100k(0.5%) , 300k(0.5%)

表 6.2: 市街地の走行を想定した設定

タスク名	命令数（発生確率）
画面描写	620k(90%) , 610k(5%) , 600k(5%)
位置情報	10k(97%) , 25k(3%)
再計算	5k(97%) , 100k(2.5%) , 500k(0.5%)

表 6.3: 住宅街の走行を想定した設定

以上のタスクを先行研究の手法で実行した場合と，提案手法を適用して実行した場合において，消費電力で比較する．

表 6.4: 評価環境

Clock Frequency	
High-end Core	2.8GHz
Low-end Core	700MHz
Target task	
タスク周期	0.5ms

シミュレータはタスク実行のシミュレーションを行う．表 6.4 にシミュレータの想定する動作環境の詳細を示す．1 ループ毎に 1 サイクルあたりの実行命令数 (IPC:Instruction Per Cycle) をタスクの残り処理量から減算することで，1 ループを 1 サイクルの処理として扱う．そして，タスクの残り処理量が 0 になる毎に次のタスクを実行し，すべてのタスクの残り処理量が 0 になった際のハイエンドコアとローエンドコアの実行サイクル数から消費電力を求める．デッドラインと全タスクの WCET との余裕度による性能の変化を測定するため，初めは WCET に対して余裕のあるデッドラインを設定し，そこからデッドラインを 5%刻みで，全タスクの WCET の合計とデッドラインが等しくなる 25%まで減らして評価を行う．コア切り替えに伴うオーバーヘッドは 1000 分の 1 周期とする．この値は先行研究で使われている値であり，本提案手法の実装に伴うスケジュー

リング機構への追加は優先度の下位ビット部の追加のみであり，処理量の増加は無いため妥当である．また，チェックポイント数が少ない場合の性質を評価するため，先行研究で用いられていたチェックポイント数が 100 の場合に加えて，チェックポイント数が 10 の場合にも評価を行った．チェックポイント数が 500，1000，5000，10000 の場合のシミュレーションも行ったが，性能の特性に大きな変化は見られなかったためチェックポイント数は 10 と 100 の場合において評価を行った．以上の条件で 1 万回の試行を行い，消費電力の平均を求める．

消費電力は，各コアで動作したクロック数に周波数の逆数を乗算して実行時間を算出し，その値に各コアの平均ワット数を乗算することで計算する．平均ワット数は HMP における消費電力削減についての研究 [6] 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 [7]．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．

6.2　評価結果

評価結果を図 6.7 に示す．実線が提案手法を併用した場合，破線が先行研究のみの場合の消費電力である．

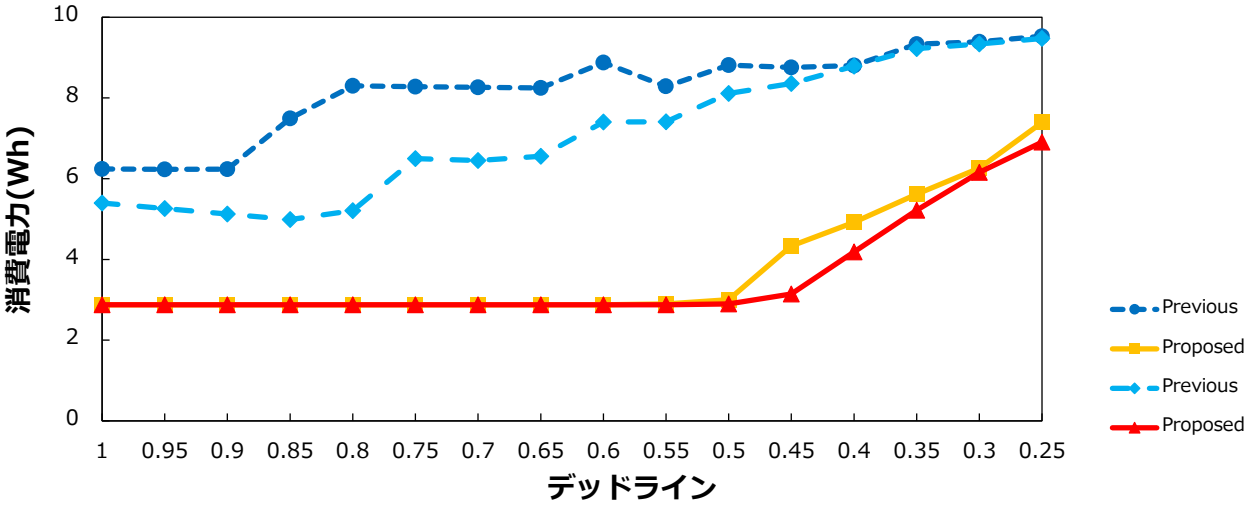

図 6.7: 消費電力の評価

評価の結果，全てのデッドライン余裕度において提案手法を併用した場合のほうが消費電力が少ないことが確認でき，最大 68%，平均 52%の消費電力の削減に成功した．特にチェックポイント数の少ない場合に消費電力の削減に大きく成功している．また，チェックポイント数を減らした場合に発生する，消費電力の増大を抑える事に成功していることが確認できた．

7　おわりに

本研究では，デッドライン制約を満たせない場合がある問題を解決し，実行時間の期待値を用いたリアルタイム OS のスケジューリング手法の提案を行った．これにより，タスク開始直後からハイエンドコアで実行しなければデッドライン制約を満たせない場合でもデッドライン制約を満たせるようになった．また，大きな徒労時間の周期後半での発生を削減するようにタスクの処理順序を改良することで，大きな余剰時間の発生を抑制することに成功し，消費電力を最大で 68%，平均 52%削減できた．

今後の展望として，本研究では問題の簡単化のために同時に動作するコアを高々1 つとしたが，複数コアを同時に動作させるスケジューリングへの拡張による高性能化が挙げられる．

謝辞

本研究の機会を与えてくださった近藤利夫教授，本研究を進めるに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沢祐樹研究員に深く感謝致します．また，本研究を行うに当たって様々なご助力，ご助言していただいたコンピュータアーキテクチャ研究室の皆様にも，心から感謝致します．

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] 嶋谷 知，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改良,2013，卒業論文

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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 Scaling. In Proceedings of the 8th International Symposium on High-Performance Computer Architecture,2002

[8] http://itpro.nikkeibp.co.jp/article/COLUMN/20050907 /220734/，参照 2．6，2018