

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

畑中 智貴^{1,a)} 中田 尚¹ 中村 宏¹

概要：動的変動タスクを扱う周期実行システムにおいては、ヘテロジニアスマルチコアプロセッサを用いて、タスクの負荷要求に応じて処理するコアを使い分けることが、省電力化のために有効である。本研究では、デッドラインまでの余裕がなくなれば、性能を上げ、余裕が生まれると、性能を下げる、というように動的に稼動コアを変更する手法を提案する。稼動コアの変更タイミングとタスクの実行時間変動確率から、稼動コアが変更される確率を求めることで、消費エネルギーの期待値を評価し、最適な稼動コアの変更タイミングを導く。提案手法と既存手法の消費エネルギーの期待値を比較すると、平均 60%、最大 76%削減されるという結果が得られた。

1. 序論

近年、組込みシステムの高性能化、小型化により、バッテリー駆動の組込みシステムが増加している。バッテリー駆動のシステムにとって、バッテリーの寿命はシステムの利便性に直結するため、要求される性能を満たした上で、消費エネルギーを削減することは、重要な課題である。

システムによって多少の差異はあるが、組込みシステムにおける消費エネルギーは、概して、プロセッサによって消費されるエネルギーが大部分である。したがって、省エネルギー化を図るためには、プロセッサにおける消費エネルギーを削減するのが効果的である。

プロセッサの性能と消費エネルギーの間にはトレードオフの関係があり、高性能なプロセッサを使用するほど、消費エネルギーは増加する。近年、性能要求の増加と、製造単価の減少に伴って、組込みシステムにおいても、マルチコアプロセッサが広く使われるようになった。特に、異種のコアが混在するヘテロジニアスマルチコアプロセッサを用いると、タスクの負荷要求に応じてコアへの割り当てを選択することができ、エネルギー効率を改善させることができる。

本研究では、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（動的変動タスク）を扱う。また、入力データは周期的に訪れるものとし（周期実行システム）、入力周期よりもデッドラインが長いシステムを対象とする。

実行を行うコアの候補が複数あり、また、デッドラインが入力周期よりも長い場合は、どのコアでいつ開始させるか、の選択肢が増えるため、消費エネルギーが最小となるようなスケジューリングを求めるのは難しく、搭載するプロ

セッサの選択も合わせると、最適なハードウェア構成とスケジューリングの組み合わせを導くのは困難を極める。

以上の問題に対し、本研究では、動的変動タスクのスケジューリングにあたり、デッドラインまでの余裕時間に応じて、実行中に稼動コア変更を行う、という手法を提案する。また、タスクの変動確率から求まる稼動コアの変更確率を用いた、消費エネルギー期待値の評価方法を提案し、最適なスケジューリングを導く。本手法を用いることで、プロセッサの消費エネルギー期待値を最小化するようなスケジューリングを実現し、また同時に、ヘテロジニアスマルチコアプロセッサを含むプロセッサ候補の中から、最適なハードウェア構成を選択することが可能となる。

2. 組込みシステムの省電力化手法

2.1 対象システム

本研究で対象とするハードウェア構成は、マルチコア、特にヘテロジニアスマルチコアプロセッサを想定する。組込みシステムでは、あらかじめ扱うタスクがわかっており、タスクの特性によって、実行するコアをうまく使い分けることで、性能効率の向上が期待できるからである。本研究では、同時に稼動するコアは1つのみとする。また、入出力データが読み書きされるメモリは各コアからアクセスできるものとする。

前述の通り、本研究では周期実行システムのうち、デッドラインが入力周期よりも十分長いシステムを扱う。一つの出力には一つの入力データしか用いない、単入力単出力系を対象とし、処理内容、処理時間は直前までの処理によって影響を受けないものとする。また、入力データの周期は一定であるとする。

一般に、タスクは、それ以上分割できない処理（サブタスク）をノードとしたタスクグラフを用いて表現できる。

¹ 東京大学
7-3-1 Hongo, Bunkyo, Tokyo 113-8656, Japan
^{a)} hatanaka@hal.ipc.i.u-tokyo.ac.jp

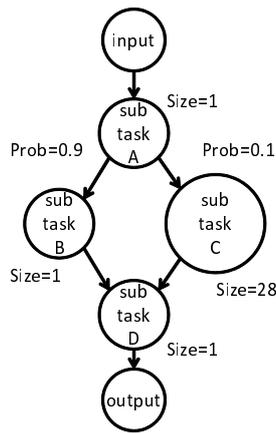


図 1 タスクグラフ例

ノードの大きさはタスクサイズ、有向エッジはサブタスクの順序関係を表している。サブタスクには、入力データによって処理内容、処理時間が変化するものが存在する。このようなサブタスクを動的である、といい、動的なサブタスクを含むタスクを動的変動タスクという。これに対して、処理時間が入力データによらず一定であるサブタスクを静的である、という。動的なサブタスクは実行完了までに実行時間を知ることはできないとする。本研究では、動的なサブタスクの実行時間の変動確率は既知のパラメータとして扱う。動的なサブタスクは、図 1 のように、タスクグラフでは分岐を用いて表現される。変動する実行時間の数だけ分岐数を設け、いずれかのパスが選択される、とすれば、静的なサブタスクのみで表すことができ、各分岐点での分岐確率は、実行時間の変動確率に一致する。本研究では、並列実行可能なタスクは扱わず、タスクグラフにおける枝分かれは、いずれかのパスのみが選択され実行されることに注意する。

2.2 消費エネルギーの削減技術

2.2.1 消費エネルギーモデル

一般に、プロセッサの消費エネルギーは、以下の式で表されることが知られている [5].

$$E_{proc} = \alpha_1 T_1 C V^2 f + T_2 V I_{leak} \quad (1)$$

第 1 項は、タスクを実行する際に消費されるダイナミックエネルギーを表す。ただし、式中の T_1 はタスクの実行時間、 C は回路の負荷容量、 V は電源電圧、 f は動作周波数、 α_1 は比例定数を表す。プロセッサのダイナミックエネルギーは、タスクの実行時間、電源電圧、動作周波数に対して、正の相関がある。動作周波数はプロセッサの性能を表しているため、プロセッサの性能と消費ダイナミックエネルギーにはトレードオフが存在する。より高性能なプロセッサを用いるほど、消費されるダイナミックエネルギーは大きくなる。

また、第 2 項は、スタティックエネルギーを表す。スタティックエネルギーとは、回路内にリーク電流が流れることにより消費されるエネルギーであり、動作/非動作に関わらず、電源が入っているときには常に消費される。ただし、式中の T_2 はプロセッサが稼動する時間、 V は電源電圧、

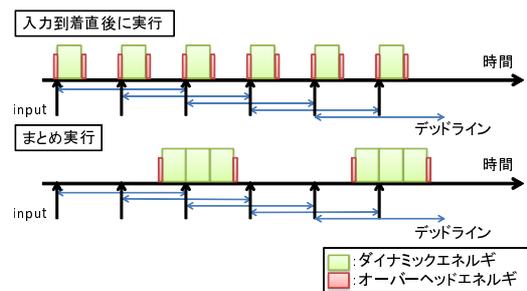


図 2 まとめ実行

I_{leak} はリーク電流を表す。リーク電流値は回路の微細化につれ増加するため、プロセッサが高性能であるほど、スタティックエネルギーは増加する。

おおまかには、プロセッサの面積は性能の 2 乗に比例し [3]、スタティックエネルギーは面積に比例することから、高性能なコアほどエネルギー効率は悪くなる。一方、マルチコアを用いると、面積は性能に比例するので、適切に並列化できれば、同性能のシングルコアと比較して、1 タスクあたりのエネルギー消費を抑えられる。

2.2.2 DPM

プロセッサの消費スタティックエネルギーを削減する方法の 1 つに DPM (Dynamic Power Management) が存在する [4]。DPM は、プロセッサの非動作時に、流れる電流を遮断することによって、スタティックエネルギーを削減する手法である。電流が遮断されている間は、プロセッサは処理を行うことができない。以降、本研究では通常の動作が可能な状態をアクティブ状態と呼び、電流が遮断されて、処理のできない状態を省電力状態と呼ぶ。状態の遷移には、オーバーヘッドエネルギーが発生するため、期待できるスタティックエネルギーの削減量と比較し、遷移を行うべきか判断する必要がある。

2.3 既存スケジューリング

タスクスケジューリングに関する先行研究のうち、本研究と関連の深い、デッドラインが十分長い場合のスケジューリング及び動的変動タスクを含むスケジューリングを紹介する。

2.3.1 デッドラインが十分長い場合のスケジューリング

デッドラインが長い場合のスケジューリングとして、タスクのまとめ実行が提案されている [1]。まとめ実行とは、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直後よりも遅らせ、後続の複数のタスクと連続して実行する、という手法である。図 2 からわかるように、まとめ実行を行うことで、アクティブ状態と省電力状態の遷移回数を減らすことができ、オーバーヘッドエネルギーの削減につながる。

2.3.2 動的変動タスクのスケジューリング

動的変動タスクを扱うスケジューリングとして、DVFS (Dynamic Voltage and Frequency Scaling) を用いた手法が提案されている [2]。具体的には、後続のサブタスクの実行時間変動確率を考慮した上で、最悪パターンを実行す

表 1 入力変数

変数	意味
M	タスクの変動パターン数
$S(i)$	パターン i のタスクサイズ
$p(i)$	パターン i が実行される確率
I	入力周期
d	デッドライン
$et(i, j)$	コア j でパターン i を実行するのに要する時間
$E_d(i, j)$	コア j でパターン i を実行するのに要するダイナミックエネルギー
$P_{Sa}(j)$	コア j のアクティブ状態のスタティック電力
$P_{Ss}(j)$	コア j の省電力状態のスタティック電力
$E_{ov}(j)$	コア j のアクティブ状態から省電力状態に移行するのに要するオーバーヘッドエネルギー

表 2 各コアの実行時間

コア	サブタスク 1 の実行時間	サブタスク 2 の実行時間
コア 1	0.3	3.6
コア 2	0.15	1.8
コア 3	0.1	1.2
コア 4	0.075	0.9

したときの実行時間 $et(i, j)$, 実行に要するダイナミックエネルギー $E_d(i, j)$, コア j のアクティブ状態時スタティック電力 $P_{Sa}(j)$, 省電力状態時スタティック電力 $P_{Ss}(j)$, アクティブ状態から省電力状態に移行するのに要するオーバーヘッドエネルギー $E_{ov}(j)$ はすべて事前に得られるものとし、状態の移行に要する時間のオーバーヘッドは無視できるものとする。

確率的に実行時間が変動するタスクを扱うため、目的関数は、「プロセッサの消費エネルギーの期待値」、制約条件は、「すべてのタスクがデッドラインを満たす」となる。

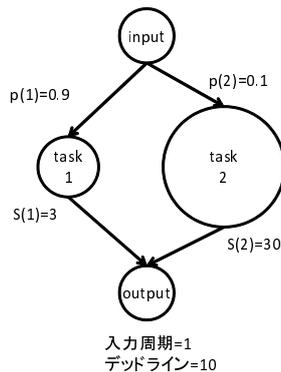


図 3 変動タスク例

ることになってもデッドライン制約を満たし、かつ、消費エネルギーの期待値が最小となるように動作周波数を設定し、サブタスクが完了する度に、デッドラインまでの余裕時間に応じて、動作周波数を変化させる、という手法である。この手法において、デッドラインは入力周期に一致しており、制御は1周期単位で行われる。また、ダイナミックエネルギーのみを削減する手法であるため、効果は限定的である。

3. 提案スケジューリング

3.1 問題設定

本節では、設計時に既知のパラメタを列挙し、本研究が掲げる問題の目的関数と制約条件を述べる。入力変数は、表 1 にまとめた。

タスクに関するパラメタは、データの入力周期 I 、デッドライン d 、変動パターン数 M 、また各タスクパターンの ID を $i = 1, 2, \dots, M$ として、変動確率 $p(i)$ 、タスクサイズ $S(i)$ が与えられる。変動パターン数は、タスクグラフにおけるパスの通り方の総数を表し、変動確率は、各パスを通る確率を表す。タスクサイズは、各パターンを実行したときの命令数に相当するパラメタである。図 1 の例は、図 3 のように変換でき、 $I = 1$, $d = 10$, $M = 2$, $p(1) = 0.9$, $p(2) = 0.1$, $S(1) = 3$, $S(2) = 30$ となる。

タスクのパターン同様、コアにも性能が低いものから順に、ID を $j = 1, 2, \dots$ とする。コア j でパターン i を実行

3.2 動的省電力スケジューリング

3.2.1 概要

本研究が提案するスケジューリングでは、タスクのまとめ実行を採用する。タスクはまとめられるだけまとめて実行するものとし、タスク実行終了時に他の実行可能なタスクがある場合は、続けて実行する。

実行時に用いられるコアの候補は以下のように選択する。最悪のパターンを続けて実行することになっても、デッドラインは守らなければならないため、最悪実行時間が入力周期を下回るような性能を持つコアが1つ必要であり、これが最大性能のコアとなる。また、最小性能のコアは、いずれのパターンを実行しても実行時間がデッドラインを超えないコアのうち、最も性能の低いコアとする。これよりも低い性能のコアでは、最悪パターンを実行したときに、必ずデッドラインを超えてしまうからである。最大性能のコア以外は最悪実行時間が入力周期を超えていることに注意する。

図 3 のような、高確率で小タスク、低確率で大タスクとなるような動的変動タスクを表 2 中のコア候補を用いて処理するとき、コア 4 のみを用いれば、入力周期 1 に対し、最悪実行時間は 0.9 であるから、必ずタスクをデッドライン以内に終わらせることができるが、この例のように、小タスクを実行する確率が高い場合には、オーバースペックとなることが考えられる。しかし、コア 4 以外のコアでは、最悪実行時間が入力周期よりも長い場合、最悪パターンが続くとデッドラインを超えるおそれが発生する。したがって、必要に応じて実行中に稼動コアを変更させた方が消費エネルギーを削減できると考えられる。

稼動コアの変更は、実行中におけるデッドラインまでの余裕時間に応じて行うこととする。つまり、余裕があるときは、低性能のコアで実行し、余裕がなくなると、高性能のコアで実行する。したがって、解決すべき問題は、デッドラインまでの余裕時間がどの値を超えたら稼動コアの性能を下げ、どの値を下回ったら稼動コアの性能を上げるか、

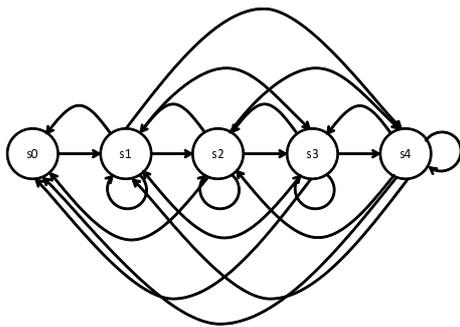


図 4 稼動コア変更の状態遷移図による表現 (N=4)

という稼動コアの変更点を決定する問題に帰着する。本研究の提案は、最適な稼動コアの変更点を決定することで、消費エネルギーの期待値を最小化しようというものである。

3.2.2 定式化と求解

目的関数の定式化にあたり、表 3 に示すパラメタを導入する。前節で述べた手法に従い、選択されたコア候補の総数を N とする。デッドラインまでの余裕時間に応じて実行中に稼動コアを変化させるが、実行中の余裕時間の計算は、1 タスクが終了する度に行うこととする。計算された余裕時間が変更点を越えていたとき、稼動コアの変更を行う。稼動コア変更は段階的な変更を想定し、主に性能が隣り合うコアへの変更を考えるが、性能を一気に変更することも許す。

ここで、稼動しているコアを一つの状態、1 タスクの実行を状態遷移として考えると、稼動コアの変更は、図 4 のように $N + 1$ 状態の状態遷移図を用いて表すことができる。ただし、どのコアも稼動していない状態を s_0 として追加し、コア j が稼動している状態を、状態 s_j とした。図 4 中の状態 s_0 からの遷移が状態 s_1 に限られるのは、まとめ実行を開始するコアをコア 1 に設定しているためである。まとめ実行を開始するコアは任意に選択することができる。

このように、稼動コアの変更を状態遷移図と捉えると、各状態間の遷移確率行列 P を定義できる。 P の (i, j) 成分は、状態 s_i から状態 s_j に遷移する確率、すなわち、コア i 稼動時に 1 タスク実行後、コア j に変更する確率を表す。 P は $(N + 1) \times (N + 1)$ の行列であり、行和は 1 となる。

また、遷移確率行列 P に対して、定常分布 π を定義することができる。定常分布 π は $1 \times (N + 1)$ のベクトルで、 $\pi P = \pi$ を満たすような分布であり、各状態に平均してどの程度分布しているかを総和が 1 になる割合として表したものと考えることができる。

加えて、まとめ実行開始から終了するまでに要する平均時間を、平均アクティブ時間 t_{active} 、まとめ実行終了から次のまとめ実行を開始するまでの平均時間を、平均スリープ時間 t_{sleep} とする。

前節で定義された目的関数は、単に消費エネルギーの期待値となっており、タスクの実行を限りなく繰り返したとき、無限大に発散してしまう。そこで、入力周期を時間幅として区切り、時間幅内で消費されるエネルギーの期待値を目的関数として定義し直す。遷移確率行列 P 、定常分布 π 、平

表 3 定式化にあたり導入したパラメタ

変数	意味
N	稼動コア候補数
P	遷移確率行列
π	P に従う定常分布
t_{active}	平均アクティブ時間
t_{sleep}	平均スリープ時間
E_D	1 タスクあたりの各状態の平均ダイナミックエネルギー
E_{Sa}	1 タスクあたりの各状態の平均アクティブ時スタティックエネルギー
E_{Ss}	1 入力周期あたりの各状態の平均省電力状態時スタティックエネルギー
E_{OV}	各状態から各状態へ遷移するのに要するオーバーヘッドエネルギー
x_{up}^i	状態 s_i から状態 s_{i+1} への変更点
x_{down}^i	状態 s_{i+1} から状態 s_i への変更点
s_*	まとめ実行を開始する状態

均アクティブ時間 t_{active} 、平均スリープ時間 t_{sleep} を用いて、目的関数 J をパラメタ表示すると、以下のように表せる。

$$J = \pi E_D + \pi E_{Sa} + \left\| \pi (P * E_{OV}) \right\| + \frac{t_{sleep}}{t_{active} + t_{sleep}} E_{Ss} \quad (2)$$

ただし、 $*$ は行列の要素ごとの積を表し、 $\|\cdot\|$ は要素の総和を表す。 E_D 、 E_{Sa} は、ともに $(N + 1) \times 1$ のベクトルであり、第 i 成分は、それぞれ、状態 s_i で 1 タスクを行ったときの平均ダイナミックエネルギー、平均スタティックエネルギーを表す。 E_{OV} は $(N + 1) \times (N + 1)$ の行列であり、 (i, j) 成分は、状態 s_i から状態 s_j に遷移するのに要するオーバーヘッドエネルギーを表す。また、 E_{Ss} は $(N + 1) \times 1$ のベクトルであり、第 i 成分は、状態 s_i において 1 入力周期で消費される省電力状態時スタティックエネルギーを表す。したがって、各項は順に、1 タスクあたりの平均ダイナミックエネルギー、1 タスクあたりの平均アクティブ時スタティックエネルギー、1 遷移あたりの平均オーバーヘッドエネルギー、1 入力周期あたりの省電力状態時スタティックエネルギーを表している。

定常分布及び平均アクティブ/スリープ時間は遷移確率行列を用いて計算されるため、式 2 を最小にする遷移確率行列を与え、その遷移確率に従うように稼動コアを変更させれば、消費エネルギーの期待値を最小化できる。

以上をまとめると、評価関数 J を最小にするような遷移確率行列 P を与える、稼動コアの変更点を求めればよい。状態 s_i から状態 s_{i+1} への変更点を、 x_{up}^i とし、状態 s_{i+1} から状態 s_i への変更点を、 x_{down}^i とする。また、まとめ実行を開始する際に稼動させるコア、すなわち、状態 s_0 から遷移する状態を s_* とする。このとき、状態 s_i から状態 s_{i+1} への変更点 x_{up}^i と、状態 s_{i+1} から状態 s_i への変更点 x_{down}^i は、異なる値をとってもよい。一度状態 s_i に遷移すると、余裕時間が x_{up}^i から、 x_{down}^{i-1} の間は、コア i で実行し続ける。この区間を、ステージ i という。 x_{up}^0 は、まとめ

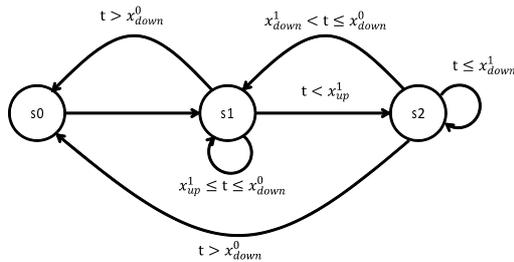


図 5 状態遷移と遷移条件

実行開始時のデッドラインまでの余裕時間を表す。また、 x_{down}^0 は、状態 s_0 への遷移であることから、まとめ実行の終了タイミングを表している。タスクはまとめられるだけまとめて実行するため、 $x_{down}^0 = d$ 、すなわち、まとめ実行の終了タイミングは余裕時間がデッドラインを超え、実行可能タスクがなくなったときである。

まとめ実行が開始されてから n 番目のタスクが終了したときのデッドラインまでの余裕時間を t_n とすると、その計算式は以下で与えられる。ただし、まとめ実行開始時のデッドラインまでの余裕時間は変更点 x_{up}^0 で表されており、まとめ実行の開始は状態 s_0 から状態 s_* への遷移である。

$$t_0 = x_{up}^0 \quad (3)$$

$$t_n = t_{n-1} + (I - et(i, j)) \quad (4)$$

まとめ実行が開始されてから n 番目のタスクが状態 s_i で実行開始されたとすると、 t_n が x_{up}^i または、 x_{down}^{i-1} を超えたとき、次の状態に遷移する。 $N = 2$ のときの遷移条件は図 5 のようになる。

稼動コア変更点 $\{x_{up}^i\}$, $\{x_{down}^i\}$ 及びまとめ実行開始コア s_* は以下の制約を満たす必要がある。

$$x_{up}^i \leq x_{down}^i \quad (i = 0, 1, \dots, N) \quad (5)$$

$$WCET_i \leq x_{up}^i \quad (i = 1, 2, \dots, N) \quad (6)$$

$$x_{up}^* \leq x_{up}^0 \leq x_{down}^{*-1} \quad (7)$$

ただし、 $WCET_i$ はコア i での最悪実行時間を表す。制約条件 5 はどの状態も定義されない余裕時間を防ぐための条件であり、制約条件 6 は、最悪パターンが訪れてもデッドラインを超えないための条件である。また、制約条件 7 は、まとめ実行の開始タイミングは、まとめ実行開始ステージ上になければならない、という条件である。

稼動コア変更点 $\{x_{up}^i\}$, $\{x_{down}^i\}$ 及び実行開始コア s_* から、タスクの変動確率を用いて、遷移確率行列 P を求めることができる。また、 P から、定常分布 π 、及び平均アクティブ時間 t_{active} 、平均スリープ時間 t_{sleep} を求めることができる。したがって、評価関数 J を計算することができ、稼動コア変更点及び実行開始コアの評価が可能となる。

具体的な稼動コア変更点及び実行開始コアの最適値は、遺伝的アルゴリズムを用いて求める。

4. 評価

4.1 実験目的

前章で提案した手法によって得られるスケジューリング

表 4 タスクに関するパラメタ

タスクに関するパラメタ	値
タスクパターン数 M	2
タスクサイズ $S(i)$	$S(1)=360, S(2)=36$
入力周期 I	0.1 ms
デッドライン d	10 ms

と、既存手法及び経験的に得られるスケジューリングの評価値を比較することで、本手法の消費エネルギー削減効果を評価する。

4.2 実験条件

比較に用いるスケジューリングは以下の 2 つを採用する。以下の手法においても、まとめ実行を行うものとし、タスク終了時に実行できるタスクがあるときは、続けて実行するものとする。

- (1) 既存手法 1：最悪パターンを想定した固定コア実行
タスクの最悪実行時間が入力周期を下回るコア 1 つのみを用いて、まとめ実行を行う。
- (2) 既存手法 2：ぎりぎりまで性能を上げず、一度上げた性能は下げない

性能を上げるタイミングは、稼動中のコアで実行し続けたら、最悪パターン実行時に、デッドラインを超えるときとし、最悪パターンを実行して間に合うコアのうち、最小性能のコアに変更する。また、一度上げた性能はまとめ実行が終了するまで落とさない。

評価に用いるタスクのパラメタは表 4、コアのパラメタは表 5 にまとめた。タスクは人物検知センサの画像処理を想定し、あらかじめ与えられた背景との差分をとることにより、変化がなければその旨を送信し、変化があれば画像処理を行うことで、人物かどうか認識し、結果を送信する、というタスクとした。したがって、タスクパターンは軽い処理と重い処理の 2 パターンとし、タスクサイズの比は 1:10 とした。また、デッドラインは 10ms、入力周期は 0.1ms に設定した。実行時間変動確率は、人物検知センサが設置される場所によって異なるため、人通りが少ない ($p(1) = 0.1, p(2) = 0.9$)、人通りが中程度 ($p(1) = 0.5, p(2) = 0.5$)、人通りが多い ($p(1) = 0.9, p(2) = 0.1$)、の 3 つの場合を想定し、それぞれ結果を求めた。プロセッサ候補数は 4 とし、本論文では一例として、文献 [1] で利用されているコアのうち最小性能のコアを MCU1、最大性能のコアを MCU2 とした。MCU2 の MCU1 に対する性能比がほぼ 4 であることから、MCU1 に対する性能比が 2 及び 3 となるような仮想的なコア (Virtual1, Virtual2) を設定し、Virtual1, Virtual2 の各パラメタは MCU1 と MCU2 の値を用いて補間した。

遺伝的アルゴリズムにおけるパラメタは、集団数を 10000 とし、世代数を 1000 とした。また、評価値の上位 1 割をエリートとして保持し、集団の 1% に突然変異が発生する、と設定した。

表 5 コアに関するパラメタ

	MCU1	Virtual1	Virtual2	MCU2
コア ID j	1	2	3	4
ダイナミック電力 $P_d(j)[W]$	0.0138	0.031	0.070	0.156
アクティブ時 スタティック電力 $P_{Sa}(j)[W]$	1.62e-3	5.84e-3	2.09e-2	7.50e-2
省電力状態時 スタティック電力 $P_{Ss}(j)[W]$	6.90e-7	2.07e-6	6.20e-6	1.86e-5
オーバーヘッド エネルギー $E_{ov}(j)[J]$	2.24e-7	1.75e-6	1.37e-5	1.07e-4

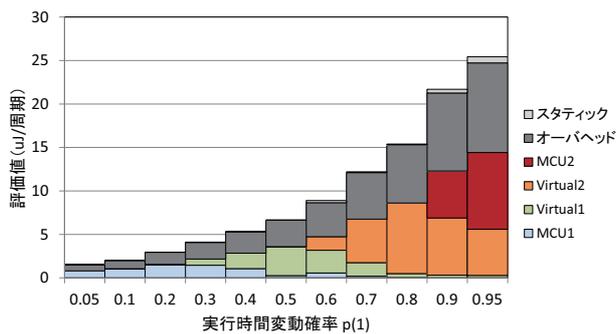


図 6 タスクの実行時間変動確率とエネルギー内訳の関係

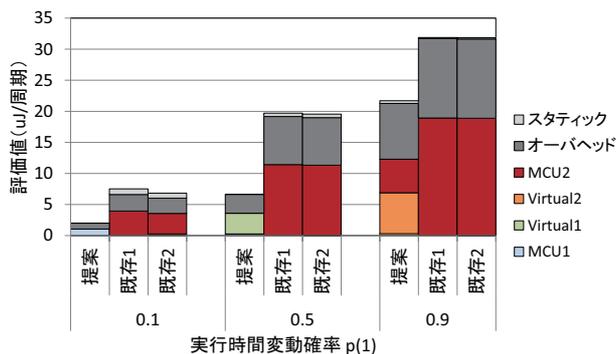


図 7 各場合の各手法の評価値とエネルギー内訳

4.3 実験結果

図 6 は、 $p(1)$ の値を細かく変化させたときの、提案手法の消費エネルギーの内訳の変化を表したグラフである。内訳については、各コアで消費されるダイナミックエネルギーとアクティブ時スタティックエネルギーの和を各コア名で表し、オーバーヘッドは総オーバーヘッドエネルギー、スタティックは省電力状態時スタティックエネルギーの総和を表している。また、各場合の提案手法と既存手法の評価値及び消費エネルギー内訳は図 7 のようになった。評価関数の定義が、1 入力周期あたりの消費エネルギーの期待値であることから、単位が [J/周期] となることに注意する。

図 6 を見ると、平均負荷に応じて、ダイナミックエネルギーを消費する主なコアが推移しているのがわかる。平均負荷が小さいとき、低性能コアの消費ダイナミックエネルギーが多くを占め、平均負荷が大きくなると、高性能コアの消費ダイナミックエネルギーが増す。また、オーバーヘッドエ

ネルギーは、高性能コアほど大きいので、高性能コアが稼動するほど大きくなっているのがわかる。

また、図 7 を見ると、いずれの場合も、評価値は提案手法が最小となった。既存手法 1 と比較すると、平均 60%、最大 78%削減されており、既存手法 2 と比較すると、平均 60%、最大 76%削減されていることがわかる。また、最悪パターンを実行する確率が低いほど、削減率が大きい、という結果となった。これは、既存手法及び経験的に得られる手法が、デッドライン制約を満たすために、常に最悪パターンを想定しているためで、最悪パターンを実行する確率が下がると、その分時間のロスが生じる確率が上がるためだと考えられる。

以上より、本提案手法を用いると、省エネルギー効果が得られること、特に、最悪パターンを実行する確率が小さい場合、著しい改善が見られることを示した。

5. 結論

本研究では、確率的に実行時間が動的変動するタスクを周期的に実行する際のプロセッサの消費エネルギーを削減するという問題に対し、まとめ実行と、デッドラインまでの余裕時間に応じた動的な稼動コア変更を組み合わせたスケジューリングを提案した。

提案手法と、既存手法及び経験的手法を比較すると、消費エネルギー期待値の削減率は平均 60%という結果になり、最悪パターンを実行する確率が低いときは、削減率が最大 76%にまで及んだ。

したがって、本提案手法を用いることで、実行時間が動的に変動するタスクを扱う組み込みシステムの消費エネルギーを効果的に削減することができる。特に、最悪パターンを実行する確率が低いときには、著しい改善が期待できる。本提案により、消費エネルギーの削減を実現し、バッテリー駆動組み込みシステムを長寿化するという課題を解決する糸口を与えた。

謝辞

本研究の一部は、NEDO「ノーマリーオフコンピューティング基盤技術開発」事業による。

参考文献

- [1] 岡本和也. “エネルギーモデルを用いたマルチコア組み込みシステムの設計支援”, 東京大学大学院情報理工学系研究科修士論文, 2013.
- [2] Jason Cong and Karthik Gururaj. “Energy Efficient Multiprocessor Task Scheduling under Input-dependent Variation” Proceedings of the Conference on Design, Automation and Test in Europe, pp. 411–416 2009.
- [3] Fred Pollack, Intel Corp. “Micro32 conference keynote” 1999.
- [4] Luca Benini, Alessandro Bogliolo, and Giovanni De Micheli. “A survey of design techniques for system level dynamic power management”. IEEE Trans. on Very Large Scale Integration Systems, Vol. 8, No. 3, 2000.
- [5] Thomas.D. Burd and Robert.W. Brodersen. “Energy efficient cmos microprocessor design”. Hawaii International Conference on System Sciences, p. 288, 1995.