Vol.2016-ARC-219 No.4
Vol.2016-SLDM-175 No.4
Vol.2016-EMB-40 No.4
2016/3/24

ヘテロジニアスマルチコア周期実行システムにおける省電力タスクスケジューリング

柳橋 宏行[1,a)]　中田 尚[1]　中村 宏[1]

概要：本稿では，ヘテロジニアスマルチコアプロセッサ環境において，周期的な複数種類のタスクが入力されたときの省電力タスクスケジューリングを提案する．これは，タスク負荷に応じて，与えられたコアの中から低性能なコア群と高性能なコア群を選択し，その 2 つをタスクの到着状況による余裕に応じて切り替えながらタスクを処理するという手法である．この手法により，周期実行システムにおいて消費電力が削減されることを示す．

キーワード：ヘテロジニアスマルチコアプロセッサ，省電力，タスクスケジューリング

1. はじめに

近年，マイクロプロセッサやメモリの製造技術の進歩によって家電製品や産業機器，医療製品に至るまで，さまざまなシステムに組込みシステムが用いられるようになった．また，組込みシステムの高性能化および小型化にともない，バッテリ駆動の組込みシステムが増加している．バッテリ駆動のシステムにとっては，システムの利便性の観点から要求される性能を満たした上で，消費エネルギを削減することは重大な課題である．

組込みシステムにおける消費エネルギは，概してプロセッサによって消費されるエネルギが大きく占めている [1]．したがって，省エネルギ化を図るためには，プロセッサにおける消費エネルギの削減が効果的である．しかし，プロセッサの性能と消費エネルギ効率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用いるほど，同一タスクに対する消費エネルギが増加する．よって，要求性能を満たすような性能をもつプロセッサを搭載した上で，適切なプロセッサを選択することで全体としての消費エネルギを削減する必要がある．

また，近年の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複数のプロセッサを搭載したマルチコアプロセッサが広く使われるようになっており，マルチコア環境における省電力スケジューリングの実現が強く求められている．

本稿で扱うタスクでは，入力データは周期的に到着する周期的なリアルタイムシステムとし（**周期実行システム**），入力周期よりもデッドラインが十分長いシステムを対象とする．また，これらのタスクが複数種類同時に入力されるマルチタスクを想定する．

一般のリアルタイムシステムにおいては，エネルギ削減の手法として **DVFS**（Dynamic Voltage and Frequency Scaling）や **DPM**（Dynamic Power Management）が用いられてきた．DVFS は，プロセッサの負荷が低いときに電圧および周波数を低下させることで，プロセッサの消費電力を削減する手法である．一方，DPM は，プロセッサがアイドル状態のときにプロセッサ全体の電源を遮断することで，エネルギを削減する手法である．ただし，電源の ON/OFF を切り替える際に，オーバーヘッドエネルギが生じてしまうことも考慮する必要がある．

本稿では，異なる種類のプロセッサが複数搭載されたヘテロジニアスマルチコアシステムにおける省電力タスクスケジューリングを提案する．これは，ヘテロジニアスマルチコアプロセッサ環境において，タスク負荷に応じて，与えられたコアの中から低性能なコア群と高性能なコア群を選択し，その 2 つを切り替えながらタスクを処理するという手法である．コア群の切り替えにはシステム全体の余裕に応じて切り替えるという手法を提案する．この手法により，周期実行システムにおいて消費電力が削減されることを示す．

[1] 東京大学大学院 情報理工学系研究科 システム情報学専攻
7-3-1 Hongo, Bunkyo, Tokyo 113–8656, Japan
[a)] yanagi@hal.ipc.i.u-tokyo.ac.jp

Vol.2016-ARC-219 No.4
Vol.2016-SLDM-175 No.4
Vol.2016-EMB-40 No.4
2016/3/24

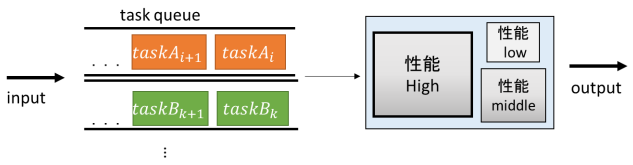

図 1 ハードウェア構成

2. 対象システムと既存省電力化手法

2.1 対象システム

2.1.1 タスクモデル

前述のように，本稿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に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は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．

本稿で扱うタスクは，一つのタスクに対し，入力から出力までの処理は一つのプロセッサが連続して行うものとする．

2.1.2 ハードウェア構成

本稿で対象とするハードウェア構成は，図 1 のような異種のプロセッサが複数搭載された，ヘテロジニアスマルチコアプロセッサを想定する．各プロセッサは同時に稼動することができるものとし，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．入力タスクは複数種類が同時に入力されるマルチタスクとし，各々のタスクは各タスクキューに入り，古いものから順に実行されるものとする．

2.2 消費エネルギモデル

一般に，プロセッサの消費エネルギは以下の式で表される [2]．

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

第 1 項はタスクの実行する際に消費されるダイナミックエネルギを表している．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 は比例定数である．動作周波数を大きくして性能を上げると，プロセッサの消費ダイナミックエネルギも増大する．第 2 項はスタティックエネルギを表している．スタティックエネルギは，回路内にリーク電流が流れることから消費されるエネルギであり，動作/非動作に関わらず電源が入っている限り常に消費される．ただし，式中の T_2 はプロセッサの稼動時間，V は電源電圧，I_{leak} はリーク電流である．リーク電流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も増加する．

同じ動作周波数で動作させるとき，プロセッサの性能はその面積の平方根に比例することが経験則（ポラックの法則）として知られており [3]，同面積で比較すると，シングルコアで動作させるときよりも性能の小さいコアを複数搭載したマルチコアで動作させたときのほうがエネルギ効率が良い．よって，マルチコアにおいて適切に並列化ができれば，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消費を抑えることができる．

2.3 DVFS

プロセッサの消費ダイナミックエネルギを削減する方法として DVFS（Dynamic Voltage and Frequency Scaling）が存在する．DVFS では，動作中のプロセッサに対し，負荷に応じて動作周波数（およびそれに付随して電圧）を変更することができる．よって，負荷が小さいときには動作周波数を小さくして供給電圧を下げることで，消費されるダイナミックエネルギを抑えることができる．ただし，制約（デッドライン）を守らなければならないことからくる制御の複雑さや，動作周波数の切り替えに伴うオーバーヘッドの影響も考慮しなければならない．

2.4 DPM

プロセッサの消費スタティックエネルギを削減する方法として DPM（Dynamic Power Management）が存在する．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でスタティックエネルギ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，本稿ではプロセッサについて通常の動作が可能な状態を**アクティブ状態**，電流が遮断されて処理できない状態を**省電力状態**と呼ぶ．こちらも状態の遷移に伴うオーバーヘッドエネルギが伴うため，期待できるスタティックエネルギの削減量と比較して遷移を行うべきか判断する必要がある．

2.5 関連研究でのスケジューリング

本稿の提案スケジューリングと同様の，デッドラインが入力周期よりも長い場合の既存研究として，タスクの実行開始を入力到着直後よりも遅らせて，後続の複数のタスクと連続して実行する，というタスクのまとめ実行が提案されている [4]．図 2 のように，まとめ実行をすることで DPM を行う際の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，赤色で示されたオーバーヘッドエネルギが 3 分の 1 に削減することができる．

また，本稿と同様のハードウェア構成，入力タスクに対し，コアを排他的に稼動させながら切り替えることで，デッドライン制約を満たしながら消費エネルギを削減するスケジューリングを提案している [5]．本稿での提案は，同様の構成に対し，コアを同時に稼動させながら実行することで

Vol.2016-ARC-219 No.4
Vol.2016-SLDM-175 No.4
Vol.2016-EMB-40 No.4
2016/3/24

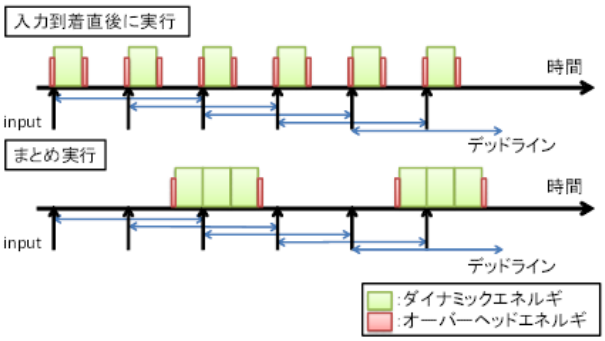

図 2 まとめ実行の様子

表 1 コア（プロセッサ）に関するパラメータ

変数	意味
M	コア数
p_c	コア c の性能（動作周波数 f と等価）
$P_d(c)$	コア c のダイナミック電力
$P_{S_a}(c)$	コア c のアクティブ状態のスタティック電力
$P(c)$	コア c のタスク実行時の電力 $P_d(c)+P_{S_a}(c)$
$P_{S_s}(c)$	コア c の省電力状態のスタティック電力
$E_{OV}(c)$	コア c の切り替え時のオーバーヘッドエネルギ
$et(j,c)$	タスク j をコア c で実行したときの実行時間
$u(c)$	コア c における CPU 使用率
$tp(C)$	コア群 C の合計性能（スループット）

表 2 タスクに関するパラメータ

変数	意味
N	入力タスクの種類数
j	タスク番号
$I(j)$	タスク j の入力周期
$d(j)$	タスク j のデッドライン
$W(j)$	タスク j のサイズ
idx_j	タスク j のインデックス
α_j	タスク j の到着オフセット（$idx_j=1$ の到着時刻）
HI	ハイパーインターバル
L	ハイパーインターバル内に入力されるタスク数
$U(j)$	タスク j のタスク負荷（$U(j)=\frac{W(j)}{I(j)}$）

より消費エネルギの削減が期待できる．

3. 提案スケジューリング

本稿で提案するスケジューリングでは，ヘテロジニアスマルチコア環境において，複数種類のタスクが周期的に到着するシステムを想定する．タスクの実行においては，文献 [5] におけるコアの排他実行とは異なり，複数のコアが同時に稼動できるものとし，全てのタスクは各々のタスクキューに入り，古いものから順番に実行されるとする．ダイナミックエネルギの削減と切り替えオーバーヘッドエネルギの削減についてそれぞれ議論することで提案スケジューリングを導く．提案スケジューリングの流れを図 3 にまとめた．

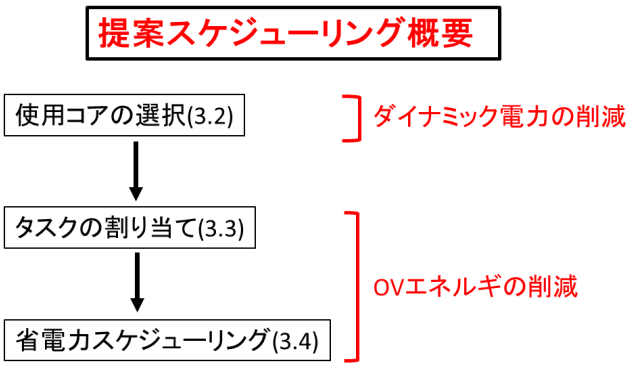

図 3 提案手法の概要

3.1 ハイパーインターバルと平均負荷性能

本稿で扱うコアとタスクモデルにおけるパラメータを表 1 と表 2 に示す．N 種類のタスクが入力されるとする．このとき，入力タスク $j(j=1,...,N)$ において，サイズを $W(j)$，到着周期を $I(j)$，デッドラインを $d(j)$ とすると，タスクは $j(W(j),I(j),d(j))$ と表される．このときタスクの入力周期の最小公倍数にあたる長い周期をハイパーインターバル（HI）と定義すると，到着タスクはこの長い周期で繰り返されると見ることができる．

$$HI = lcm(I(1),I(2),...,I(N)) \qquad (2)$$

以下，今回想定するタスクのデッドラインはこの HI よりも大きい，すなわち $\forall j, d(j) > HI$ を満たすものとする．また HI 内に到着する全タスクを HI ちょうどで処理できるコアの性能を平均負荷性能と呼ぶことにする．

3.2 使用コアの選択

ダイナミックエネルギの削減について，どのコアを選択すべきかという観点で説明する．N 種類のタスクが入力されるとき，各タスクの実行時間がデッドライン制約を満たさないコア，すなわち $et(j,c) > d(j)$ となるタスク j が存在するコアについては，性能を満たさないものとして予め除外する．今，コアが全部で M 個あったとすると，選択可能なコアの組み合わせは，全て使用しないという状態を除いて 2^M-1 通り存在する．各々の選択されたコアの集合を C_m と名づけると，各コア集合に対してコアの性能を合計したスループットを定めることができ，$tp(C_m)$ と表すことにする．ただし，コア群 C に対するスループットは $tp(C)=\sum_{c\in C} p_c$ と計算されるものとし，添え字の m はスループットの順に 1,2,... と付けるものとする．

今，横軸を性能（スループット），縦軸を消費電力として各コア群をプロットすることを考える．一般に，プロセッサにおける消費エネルギはその性能についての下に凸な増加関数となっている．しかしコア群を性能順にプロットしても，消費電力は規則的には並ばず，不規則で離散的な値をとる．そこで，まずはコア群の中から使用するコア群の

Vol.2016-ARC-219 No.4
Vol.2016-SLDM-175 No.4
Vol.2016-EMB-40 No.4
2016/3/24

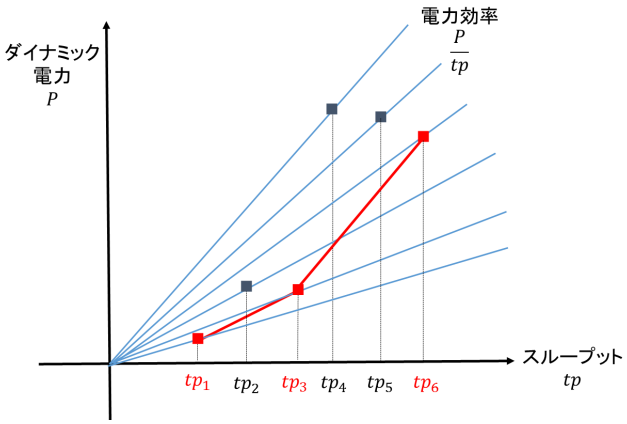

図 4 電力効率によるコア群の選択

候補を絞り込むことを考える．

コア群の候補を絞り込む際の指標となるのが，電力効率である．ここで言う電力効率とは，コア群の消費電力の和をそのコア群のスループットで割った値，すなわち性能あたりの消費電力と定義する．

$$(\text{電力効率}) = \frac{\sum_{c \in C_m} P(c)}{tp(C_m)} \tag{3}$$

この電力効率は，横軸を性能（スループット tp），縦軸を消費電力 P としてプロットした点と原点を結んだ直線の傾きにあたる．

コア群の候補の絞込みは，次のように行えばよい．まず，電力効率が最良（傾き最小）のコア群を選ぶ．そのコア群よりも低い性能のものは電力効率の観点から使用する必要がなくなるので除外する．そして残ったコア群の中から次に電力効率が良いものを選び，同様にそれよりも性能の低いものを除外し，これを繰り返していく．すると図 4 のように．選ばれたコア群を結んだ線分が，元のコア群候補同士を結んだ線分の中で最下となる組み合わせが残る．

これら絞り込まれたコア群の候補を $C'_{m'}(m' = 1, 2, ...)$ と名づけると，平均負荷性能の前後の性能を持つ 2 つのコア群 C_{low} と C_{high} を選択して使用することがダイナミック電力の観点から最適となる．なぜならば，その他の組み合わせや 3 つ以上の組み合わせで適切な比率で用いることで平均負荷性能を実現しようとすると，C_{low}，C_{high} を使用したときよりもダイナミック電力が大きくなるからである．

3.3 タスクの割り当て

使用するコア群を選択した後，タスクの割り当てを行う．今，使用するコア群 C_{low} と C_{high} に属しているコアをそれぞれ低性能のものから順に $c_m^{low}(m = 1, 2, ...)$，$c_{m'}^{high}(m' = 1, 2, ...)$ とし，N 種類のタスクをタスク負荷 $U(j)$ の順に並べ，$j = 1, 2, ..., N$ と番号付けるとする．C_{low} と C_{high} を交互に切り替えてタスクを実行し続けるが，このときタスクの割り当て方で同じ状態に留まることのできる長さが変わる．より長く留まるためには，各状態でのコア同士の CPU 使用率 $u(c)$ がなるべく等しくなっていることが望ましい．以下，そのための割り当て方を説明する．

ある状態（コア群）内での割り当て方を説明する．コア群 C に属している M' 個のコアを $c_m(m = 1, 2, .., M')$ と表記する．まず，タスクの割り当ての方針として，タスク負荷の小さいものは低性能なコアに，負荷の大きいものは高性能なコアに割り当てるものとする．これは，各タスクの HI あたりの実行時間をなるべく合わせることを意図したものである．

CPU 使用率が等しくなるような理想的なタスク負荷の分配比は性能 $p(c)$ の比率に等しい．よって各コアでの理想的なタスク負荷 $U_{ideal}(c)$ は，タスク全体の負荷 $\sum_{1 \le j \le N} U(j)$ を用いて，

$$U_{ideal}(c) = \frac{p(c) \sum_{1 \le j \le N} U(j)}{tp(C)} \tag{4}$$

と表される．よって，なるべくこの理想的なタスク負荷に合わせるために，タスク j を負荷の小さい順から $j = 1, 2, ...$ と順番に割り当てていき，$U_{ideal}(c)$ を超える直前，もしくは超えた直後をそのコアへのタスクの割り当てとすればよい．

これを性能の低いコアから順に繰り返していくと，割り当て方は各コアで $U_{ideal}(c)$ を超えるか超えないかのどちらを選択したかを考えて，$2^{M'-1}$ 通り存在する．それらについて，理想的な CPU 使用率との差を合計したものを評価関数とし，それらを最小化する割り当て方を採用する．

$$min \sum_{c \in C} \left| \frac{\sum_{1 \le j \le N} U(j)}{tp(C)} - u(c) \right| \tag{5}$$

ここで，N 個のタスクの入力周期が全て等しいときなどのように，HI 内に入力される各タスクの数が十分存在しないとき，デッドラインの最小値 $min\ d(j)$ 以下の長さまでならば HI を整数倍にして伸ばすことができる．これを用いればより細かくタスクを分配することが可能である．

以上のようにタスクを割り当てたとき，C_{high} での割り当て方に関して，以下の条件を満たすことが必要である．

$$\forall c \in C_{high},\ u(c) < 1 \tag{6}$$

これを満たさない場合はデッドラインミスをしてしまうことになり，そのコア群 C_{high} を用いることはできない．より性能の高いコア群を使用するか，もしくはそれが存在しないときはスケジューリング不可となる．

3.4 省電力スケジューリング

次に，オーバーヘッドエネルギの削減の観点から，タスクのまとめ実行およびコアの切り替え方を説明する．今，

Vol.2016-ARC-219 No.4
Vol.2016-SLDM-175 No.4
Vol.2016-EMB-40 No.4
2016/3/24

デッドラインが入力周期よりも長いタスクを想定しているので，図 2 のようにまとめ実行をすることが可能である．タスクをまとめ実行にて実行し続けるためには，C_{low} と C_{high} を切り替えながら実行する必要がある．コアを使用しないときは DPM を用いて省電力状態に移行するとし，その際の切り替え回数を減らすことを考え，なるべく低性能コア群 C_{low} と高性能コア群 C_{high} に留まり続けられるようにする．

提案スケジューリングでは低性能のコア群と高性能のコア群を切り替えながらタスクを実行する．コアの切り替えは 1 タスクの実行が終了したタイミングで行う．コアの変更は以下のタイミングで行う．

(1) 開始時はタスクをまとめ実行できるまで待機
(2)C_{low} でタスクの実行を開始
(3) 未来のタスクのデッドラインが間に合わなくなる直前に C_{high} に切り替え
(4) 未来のタスクのデッドラインの実行に隙間が生じてしまう直前に C_{low} に切り替え
(5)(2) へ戻る

3.4.1 コア切り替え条件

C_{low}, C_{high} に属しているコアの数をそれぞれ M'_{low}, M'_{high} とし，それぞれ属するコアを性能の順に $c^{low}_m (m = 1, 2, ..., M'_{low})$，$c^{high}_{m'} (m' = 1, 2, ..., M'_{high})$ とする．このとき，C_{low}, C_{high} 内に共通のコアがある，すなわち $c^{low}_m = c^{high}_{m'}$ なるコアが存在していても問題ない．ここで，タスクの分配により合わせて合計 $M'_{low} + M'_{high}$ 種類の HI が定義される．それぞれの HI を，コア c の添え字に合わせて $HI^{low}_m, HI^{high}_{m'}$ と表記する．

[実行開始時]
各々のコアの HI 分の入力タスクの C_{low} での実行時間を足し合わせ，各々のタスクの「実行開始予定時刻 - 到着時刻」の絶対値最大分の時間だけ実行開始を遅らせる．その中の最大値分待って同時に実行開始する．

[$C_{low} \rightarrow C_{high}$ への切り替え]
次のタスクを C_{low}，その次の各々のコアでの HI 分の入力タスクを C_{high} で実行すると，いずれかのタスクがデッドラインに間に合わなくなるときに切り替える (図 5)．このタイミングで切り替えないとデッドラインミスをすることになる．

[$C_{high} \rightarrow C_{low}$ への切り替え]
次のタスクを C_{high}，その次の各々のコアでの HI 分の入力タスクを C_{low} で実行するときと，実行が終了したコア c_{tmp} での HI 分の入力タスクの，いずれかのタスクが到着時間より先に実行開始してしまうときに切り替える (図 6)．このタイミングで切り替えないとアイドル時間が発生してしまうことになる．

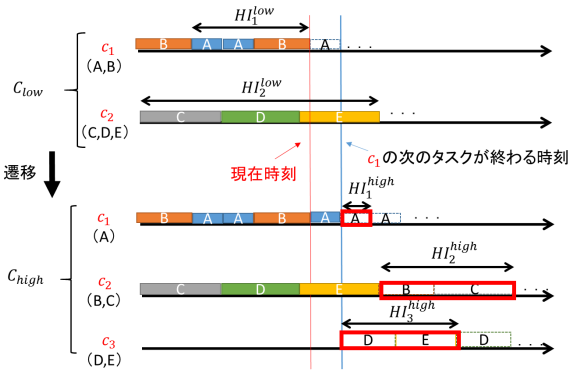

図 5 C_{low} から C_{high} への遷移判定

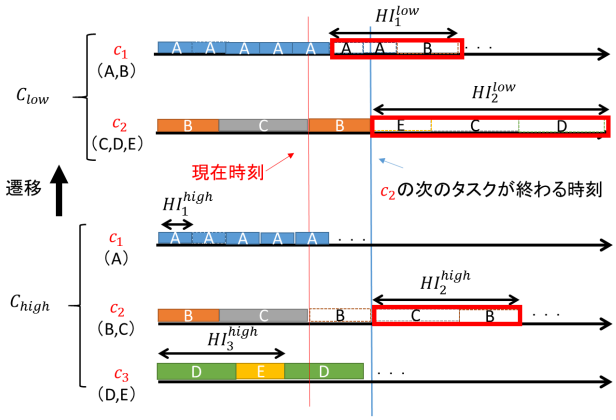

図 6 C_{high} から C_{low} への遷移判定

表 3 評価タスクに関するパラメータ

タスクに関する変数	タスク 1,2,3	タスク 4,5,6
タスク番号	$j = 1, 2, 3$	$j = 4, 5, 6$
入力周期 $I(j)$	100 ms	40 ms
デッドライン $d(j)$	250 ms	250 ms

表 4 コアに関するパラメータ

コアに関する変数	MCU1	MCU2	MCU3	MCU4
性能比	1	2	3	4
コア番号	c_1	c_2	c_3	c_4
$P_d(c) + P_{s_a}(c)$ [W]	0.0154	0.0368	0.0909	0.231
$P_{s_s}(c)$ [W]	6.90e-7	2.07e-6	6.20e-6	1.86e-5
$E_{OV}(c)$ [J]	5.10e-5	1.24e-4	2.53e-4	4.30e-4
$et(j, c)$(j=1,2,3) [ms]	2.4	1.2	0.8	0.6
$et(j, c)$(j=4,5,6) [ms]	49.8	24.9	16.6	12.45

4. 評価

本提案のスケジューリングによる消費エネルギを算出し消費電力に変換して，他の手法と比較した．評価にあたって，タスクの実行を再現し消費エネルギを積算するシミュレータを作成した．

Vol.2016-ARC-219 No.4
Vol.2016-SLDM-175 No.4
Vol.2016-EMB-40 No.4
2016/3/24

表 5　使用されたコア

手法	タスク計 4 個	タスク計 5 個	タスク計 6 個
（1）	c_2	c_3	c_4
（2）	c_2	c_3	c_4
（3）	c_{low} : c_1	c_{low} : c_2	c_{low} : c_3
	c_{high} : c_2	c_{high} : c_3	c_{high} : c_4
（4）	C_{low} : $\{c_1\}$	C_{low} : $\{c_1\}$	C_{low} : $\{c_1,c_2\}$
	C_{high} : $\{c_1,c_2\}$	C_{high} : $\{c_1,c_2\}$	C_{high} : $\{c_1,c_2,c_3\}$

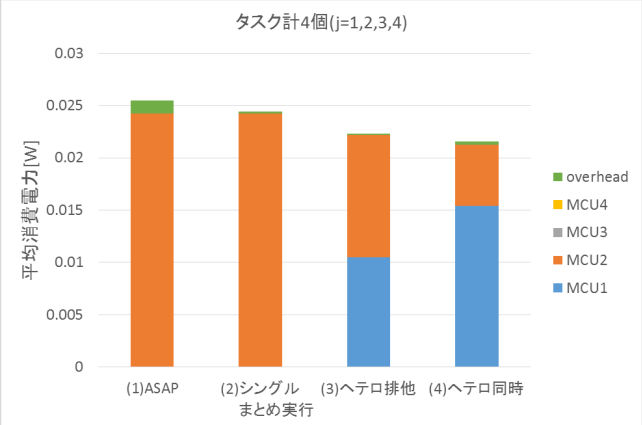

図 7　タスク計 4 個での平均消費電力の比較

4.1　実験条件

入力タスクは処理の軽いセンサタスク (j=1,2,3) と処理の重いイメージセンサタスク (j=4,5,6) の最大 6 個とし表 3 にまとめた．タスクは計 4 個 (j=1,...,4) から計 6 個 (j=1,...,6) まで変化させて与えた．コア候補は c_1〜c_4（性能比 1:2:3:4）の 4 コアとし，各パラメータを表 4 にまとめた．平均消費電力を以下 4 つのスケジューリング手法で比較した．従来手法として（1）ASAP を用いた．本稿で説明したのは（4）の手法であり，（2）は一つのコアでまとめ実行，（3）は文献 [5] で提案したコアを排他的に切り替えながら実行するスケジューリングである．

（1）ASAP：到着タスクの実行が可能となった直後に一つのコアで実行

（2）シングルまとめ実行：一つのコアでまとめ実行

（3）ヘテロ排他：2 つのコアを排他的に稼動させながら実行

（4）ヘテロ同時：2 つのコア群を切り替えながら実行

4.2　平均消費電力の比較

図 7，図 8 にタスクが計 4 個，6 個入力されたときの手法ごとの平均消費電力を示す．また，各手法において使用されたコアを表 5 に示した．エネルギの内訳において MCU1〜MCU4 はそれぞれ c_1〜c_4 を表す．タスク計 4 個，計 5 個，計 6 個いずれの場合も提案手法（2）〜（4）の順に平均消費電力が小さくなっており，（1）と（4）を比較すると，計 4 個では 15.4 %の削減，計 6 個では 65 %の削減が見込めることが分かった．

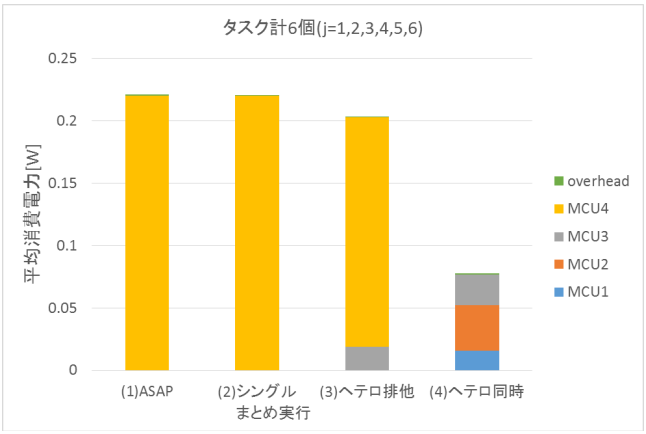

図 8　タスク計 6 個での平均消費電力の比較

5.　まとめ

本稿では，ヘテロジニアスマルチコア環境において，周期的な複数種類のタスクが入力されたときの省電力スケジューリングを提案した．低性能なコア群と高性能なコア群を選択し，切り替えながら実行することで，デッドライン制約を満たしながら平均消費電力を削減することが可能な省電力タスクスケジューリングを提案した．また，シミュレータを作成して，実際に消費エネルギの削減効果があることを確認した．

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] Victor Shnayder *et al.* “Simulating the power consumption of large-scale sensor network applications,” Proceedings of the 2nd international conference on Embedded networked sensor systems, SenSys ’04, pp. 188–200, New York, NY, USA, 2004. ACM.

[2] Thomas.D. Burd and Robert.W. Brodersen. “Energy efficient cmos microprocessor design”. Hawaii International Conference on System Sciences, p. 288, 1995.

[3] Fred Pollack, Intel Corp. “Micro32 conference keynote” 1999.

[4] T. Nakada *et al.* Design Aid of Multi-core Embedded Systems with Energy Model, IPSJ Transactions on Advanced Computing Systems, Vol.7, No.3 (ACS46), pp. 37–46, (2014)

[5] Takashi Nakada, Hiroyuki Yanagihashi *et al.* ”Energy-Efficient Continuous Task Scheduling for Near Real-time Periodic Tasks,” The 8th IEEE International Conference on Internet of Things (iThings), pp.675–681, Dec. (2015)