Vol.2015-SLDM-170 No.33
Vol.2015-EMB-36 No.33
2015/3/7

同一命令セットヘテロジニアスマルチコアのための消費エネルギーを削減するタスクスケジューリング

岩田 淳[1]　高瀬 英希[1,a)]　高木 一義[1]　高木 直史[1]

概要：同一命令セットヘテロジニアスマルチコアとは，高性能コアと高電力効率コアを排他的に動作させることで，システムの性能向上と消費エネルギー削減を両立するプロセッサアーキテクチャを指す．本稿では，組込みリアルタイムシステムにおける同一命令セットヘテロジニアスマルチコアを対象とした消費エネルギーを削減するタスクスケジューリング手法を提案する．まず，高性能コアと高電力効率コアを一対一に対応させたコアペアとし，それぞれのコアの性能比と周波数設定値から定義される正規化性能を導入する．これにより，シングルプロセッサ向け動的・電圧周波数制御（DVFS）手法によってコアペア内の動作コア切替えを実現する．提案するコアペア内の DVFS 手法は，各タスクの実行時間の平均値と最悪値の比から正規化性能を算出する Load Balancing with Average Ratio，および，デッドライン制約を保証する正規化性能を既存手法より厳密に算出する full look ahead EDF からなる．2 種類の方法で算出された正規化性能のうち高いものを選択することで，DVFS の適用においてコアペア内の周波数を平準化して消費エネルギーを最小化する．評価の結果，提案手法は，最大で 59.2%の消費エネルギーを削減できた．

1. はじめに

近年の組込みシステムには，より高い性能が求められるとともに，その消費エネルギーを削減することが重要な課題となっている．また，組込みシステムには高いリアルタイム性が要求され，システムの処理単位であるタスクの実行を，それぞれの所定の時刻であるデッドラインまでに完了する必要がある．これをデッドライン制約という．

近年，高性能化と消費エネルギー最小化の両立に貢献するプロセッサアーキテクチャとして，同一命令セットヘテロジニアスマルチコアが注目されている．このアーキテクチャでは，異なる性能と電力効率によって設計された複数のコアを持つ．各コアの命令セットは同一であるため，その差異を意識せずに動作コアを切り替えることができる．一般に，高性能コアと高電力効率コアを排他的に切り替えてタスクが実行される．負荷の大きい時は高性能コアを，そうでない時は高電力効率コアを動作させることで，性能を落とすこと無く消費エネルギーの最小化を実現する [1]．

動的電圧・周波数制御（Dynamic Voltage and Frequency Scaling，DVFS）を適用できる組込みシステムでは，コアの周波数を適切に決定することが消費エネルギー最小化を達成するための重要な課題となる．DVFS とは，コアの供給電圧および動作周波数を適切に制御する技術である [2]．CMOS 回路の消費エネルギーは，周波数の 2 乗に比例すると近似できる．ゆえに，DVFS によってコアの供給電圧と周波数を下げてタスクを実行することで，消費エネルギーが削減できる．シングルプロセッサにおいては，周波数が全タスクのデッドライン制約を保証しつつ平準化されるように DVFS を適用できる場合に，消費エネルギーが最小となる [3]．同一命令セットヘテロジニアスマルチコアでは，DVFS に加え，動作コアの適切な切替えも重要となる．

本研究では，同一命令セットヘテロジニアスマルチコアを採用した組込みリアルタイムシステムにおいて，要求される性能を保証した上での消費エネルギーの最小化を目指す．タスクの割付け対象は，高性能コアと高電力効率コアを一対一に対応させたコアペアとして扱い，コアペア内における消費エネルギー削減手法を提案する．

まず，コアペアを構成するそれぞれのコアの性能比および周波数設定値から正規化性能を定義し，コアペアテーブルを生成する．コアペアテーブルを導入することにより，シングルプロセッサ向けの DVFS によってコアペア内の動作コアの切替えを実現する．その上で，2 種類の算出方法から得られる正規化性能のうちより高いものを選択し，DVFS を適用する手法を提案する．1 つ目の算出方法は，各タスクの実行時間の平均値と最悪値の比を用いて正規化性能を算出する Load Balancing with Average Ratio

[1] 京都大学 大学院情報学研究科
[a)] takase@i.kyoto-u.ac.jp

Vol.2015-SLDM-170 No.33
Vol.2015-EMB-36 No.33
2015/3/7

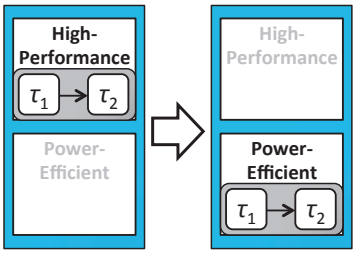

図 1 動作コアの切替えの例

（LBAR）である．2 つ目は，デッドライン制約を保証する最低の正規化性能を既存手法より厳密に算出する full look ahead EDF（flaEDF）である．以上の提案手法により，同一命令セットヘテロジニアスマルチコアのコアペア内において，デッドライン制約を保証した上での消費エネルギーの削減を実現する．なお，提案する DVFS 手法は，シングルプロセッサ向けの既存手法である look ahead EDF（laEDF） [4] にある問題点を解消しており，より高い消費エネルギー削減効果が得られる．

2. 同一命令セットヘテロジニアスマルチコア

同一命令セットヘテロジニアスマルチコアとは，同じ命令セットを持ち，性能が異なる複数のプロセッサコアを持つアーキテクチャである．本アーキテクチャでは，命令セットやデータの変換を意識することなく，タスクのコア割付けを変更できる．代表例として，ARM 社の big.LITTLE アーキテクチャ [5] や NVIDIA 社の Variable SMP[6] が挙げられる．前者については，商用プロセッサとして Samsung 社の Exynos 5 Octa やルネサスエレクトロニクス社の MP6530 がある．文献 [7] は，オンチップメモリを共有した高性能コアと高電力効率コアで構成される同一命令セットヘテロジニアスマルチコアの実チップ設計を示している．これらのプロセッサでは，適切なタスクスケジューリングによって高性能コアと高電力効率コアを排他的に動作させることで，高い性能対電力効率を実現できる．

同一命令セットヘテロジニアスマルチコアでは，高性能コアおよび高電力効率コアを対応付けてコアペアとして，動作コアの切替えが適用できる．タスクセットは，その状況に応じて，コアペア内の高性能コアと高電力効率コアのいずれかでスケジューリングされる．**図 1** は，高性能コアから高電力効率コアに動作コアを切り替える例を示している．ただし，動作コアの切替えは，デッドライン制約を保証するように行う必要がある．

3. 対象とするシステムモデル

本研究の対象とするシステムは，高性能コア $Core_{HP}$ および高電力効率コア $Core_{PE}$ で構成されるコアペア CP を持つ．$Core_{HP}$ および $Core_{PE}$ の周波数は，それぞれ F_{HP,m_0}（$m_0 = 0, 1, ..., M_{HP} - 1$，$F_{HP,0} > F_{HP,1} > \cdots > F_{HP,M_{HP}-1}$）および F_{PE,m_1}（$m_1 = 0, 1, ..., M_{PE} - 1$，$F_{PE,0} > F_{PE,1} > \cdots > F_{PE,M_{PE}-1}$）である有限個の離散値が設定できる．各周波数の設定値に対する動的な消費電力は，それぞれ，W_{HP,m_0} および W_{PE,m_1} とする．各コアの IPC は周波数に依らず，それぞれ IPC_{HP} および IPC_{PE} とする．なお，タスクの実行時間は，動作するコアの周波数に反比例するとする．なお，各コアのアイドル時における静的電力，および，DVFS の実行にかかる消費エネルギーは無視できるものととする．

タスクセット T は，I 個のタスク $\tau_0, \tau_1, ..., \tau_{I-1}$ から構成され，動的優先度ベースのリアルタイムスケジューリングである Earliest Deadline First（EDF） [8] に従って実行されるとする．すなわち，全タスクは周期タスクであり，τ_i のリリース周期 P_i と相対デッドライン D_i は等しい．絶対デッドライン時刻 d_i は，τ_i のリリース毎に D_i を加算した値に更新される．各タスクは独立して動作し，あるタスクのリリース時に最も d_i が小さい τ_i が最高優先度となる．なお，DVFS 手法の説明の都合上，添字 i は，その時点で優先度が高いものほど小さい値をあらわすとする．つまり，$d_0 \leq d_1 \leq \cdots \leq d_{i+1}$ を満たす．最悪実行時間 C_i は，$Core_{HP}$ の最高周波数 $F_{HP,0}$ で実行する場合の最長の実行時間であり，既知とする．τ_i の負荷 u_i は $u_i = \frac{C_i}{P_i}$，タスクセット T の負荷 U は $U = \sum_i u_i$ と定義される．

4. 既存手法 look ahead EDF とその問題点

EDF を採用した組込みシステムに適用できる look ahead EDF（laEDF） [4] は，時刻 t において，最高優先度タスクに後続するタスクが最高の周波数で実行されると仮定して，デッドライン制約を保証する範囲で最低の周波数を算出する DVFS 手法である．周波数の算出のため，ある区間におけるコアの実行容量およびタスクの仕事量を定義する．前者は，その区間の時間と最高周波数の積である．後者は，その区間におけるタスクの実行時間と設定される周波数の積である．

laEDF は，タスクのリリース時およびディスパッチ時に最高優先度タスクの周波数を算出して DVFS を適用する．**図 2** に laEDF による周波数 F_{laEDF} の算出手順を，**図 3** に適用例を示す．laEDF では，現在時刻 t における τ_i の残り最悪実行時間 c_rem_i を用いて予約される仕事量 s を求める．低優先度のタスクから順に，その最悪時の仕事量を直近のデッドライン時刻 d_0 から最低優先度タスクのデッドライン時刻 d_{I-1} までの実行容量に予約していく．これにより，d_0 までに予約される仕事量を小さくし，t で設定する周波数を低く抑える．詳細は文献 [4] を参照されたい．

laEDF の問題点について議論する．まず，laEDF は，タスクセットの情報のみから周波数を算出するアルゴリズムであり，コア毎の性能および電力効率の差異については考慮していない．このため，同一命令セットヘテロジニアス

Input: t, c_rem_i, d_i, u_i, U
Output: F_{laEDF}
1: $U' \Leftarrow U$
2: $s = 0$
3: **for** $i = I - 1$ to 0 **do**
4: $\quad U' \Leftarrow U' - u_i$
5: $\quad x \Leftarrow \max\{0, c\_rem_i - (1 - U') \cdot (d_i - d_0)\}$
6: $\quad U' \Leftarrow U' + (c\_rem_i - x)/(d_i - d_0)$
7: $\quad s \Leftarrow s + x$
8: **end for**
9: $F_{laEDF} \Leftarrow F_0 \cdot s/(d_0 - t)$

図 2 laEDF による周波数の算出

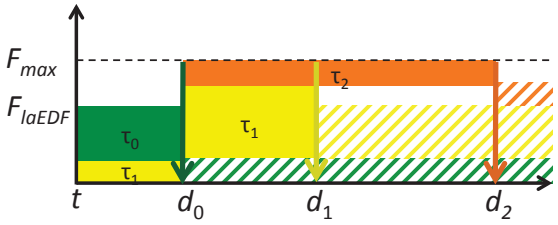

図 3 laEDF の適用例

マルチコアにおけるコアペア内での動作コアの切替えには対応できない．次に，laEDF は，d_0 以前の周波数のみに着目しているため，周波数を過剰に引き下げた後に，d_0 以降で周波数を大きく上げなければならない場合が生じうる．例えば，最高優先度タスクが最悪実行時間で実行されると，次にディスパッチされるタスクの周波数は F_{max} に設定する必要がある．これは，laEDF は最悪実行時間によるタスクの仕事量のみに着目しているためである．さらに，laEDF は，デッドライン制約を保証するには余裕がある，必要以上に高い周波数が算出されることがある．これは，周波数算出の簡単化のために，直近のデッドライン以降で仕事量を予約しない部分が生じるためである．最後に，laEDF は，タスクのリリース時とディスパッチ時のいずれでも，仕事量の予約と周波数の算出が常に行われる必要がある．このため，DVFS の適用にかかる時間オーバヘッドが大きくなることが考えられる．

5. コアペア内の DVFS 手法

本節では，まず，コアペアの正規化性能を定義し，シングルプロセッサ向けの DVFS によって同一命令セットヘテロジニアスマルチコアにおける動作コアを選択できる手法を提案する．その後，前節で紹介した laEDF の問題点を改善したコアペア内の DVFS 手法を提案する．提案手法は，Load Balancing with Average Ratio（LBAR）および full laEDF（flaEDF）という 2 種類の方法で算出された正規化性能のうち高いほうを採用して DVFS を適用する．これにより，デッドライン制約を保証した上でより高い消費エネルギー削減効果を得る．

表 1 コアペアテーブルの例

NF_n	PW_n	コアの種類	周波数
1.00	2300	0	0
0.80	1500	0	1
0.55	1200	0	2
0.30	1000	1	0
0.15	750	1	1

5.1 コアペアテーブル

コアペアの正規化性能 NF を定義する．これは，高性能コアの最高周波数における性能を 1 としたときの，コアペア内の各コアの各周波数設定値における性能を正規化した値であり，以下のように計算できる．

$$NF_{HP,m_0} = \frac{F_{HP,m_0}}{F_{HP,0}}, \quad NF_{PE,m_1} = \frac{IPC_{PE}}{IPC_{HP}} \cdot \frac{F_{PE,m_1}}{F_{HP,0}} \quad (1)$$

NF から，各コアの性能対消費電力 PW が定義される．

$$PW_{HP,m_0} = \frac{W_{HP,m_0}}{NF_{HP,m_0}}, \quad PW_{PE,m_1} = \frac{W_{PE,m_1}}{NF_{PE,m_1}} \quad (2)$$

以上から，高性能コアと高電力効率コアの情報を正規化性能によって 1 つにまとめた，コアペアテーブルを生成する．**表 1** に，コアペアテーブルの例を示す．生成時には，各行を正規化性能で降順に並べ，隣り合う行について NF が低くかつ PW が大きいものは，テーブルから除外する．つまり，コアペアテーブルに格納される情報は，正規化性能 NF_n および性能対消費電力 PW_n（$n = 0, 1, ..., N-1$，$NF_0 = 1 > NF_1 > \cdots > NF_{N-1}$，$PW_0 > PW_1 > \cdots > PW_{N-1}$）となる．これらに加え，コアペア内の動作コアおよびその動作コアにおける周波数の識別子を表す情報が格納される．例えば，表 1 の最後の行は，高電力効率コアで周波数 $F_{PE,1}$ を表す．

正規化性能が定義されたコアペアテーブルを利用することで，DVFS の枠組みでコアペア内の動作コアを切り替えることができる．DVFS 手法で算出される周波数は，高性能コアの最高周波数の性能で正規化すれば，正規化性能と同等に扱える．そして，正規化された値以上で，かつ，コアペアテーブルにある最低の NF_n となる行を選択する．その行の値から，動作コアの種類および周波数を設定する．

5.2 Load Balancing with Average Ratio

LBAR は，隣接するデッドライン間の実行容量に対して，各タスクの平均の仕事量を適切に予約し，全体の負荷を平準化できる正規化性能を算出する手法である．LBAR では，優先度の高いタスクからその平均の仕事量を予約していく．優先度の高いタスクは，デッドラインが近いため，仕事量を予約できる実行容量が小さい．一方で，優先度の低いタスクは，仕事量を予約できる区間が大きいため，負荷を平準化するための調整が容易となる．

平均の負荷 u_i^{av} は，あらかじめ得た平均実行時間と最悪

Vol.2015-SLDM-170 No.33
Vol.2015-EMB-36 No.33
2015/3/7

Input: t, c_rem_i, d_i, u_i, AR_i, NF_n
Output: NF_{LBAR}

```
for i = 0 to I − 1 do
    for n = 0 to N − 1 do
        load_th_{i,n} ⇐ NF_n · (d_i − d_{i−1})
    end for
end for
U^av ⇐ 0
n' ⇐ N − 1
for i = 0 to I − 1 do
    load_i ⇐ U^av · (d_i − d_{i−1})
    c_rem^av ⇐ AR_i · c_rem_i
    while n' ≥ 0 ∧ c_rem^av > 0 do
        for i' = i to 0 do
            if load_th_{i',n'} − load_{i'} ≥ c_rem^av then
                load_{i'} ⇐ load_{i'} + c_rem^av
                c_rem^av ⇐ 0
                break
            else
                c_rem^av ⇐ c_rem^av − (load_th_{i',n'} − load_{i'})
                load_{i'} ⇐ load_th_{i',n'}
                if i' = 0 then
                    n' ⇐ n' − 1
                end if
            end if
        end for
    end while
    U^av ⇐ U^av + AR_i · u_i
end for
NF_LBAR ⇐ NF_{n'}
```

図 4　LBAR による正規化性能の算出

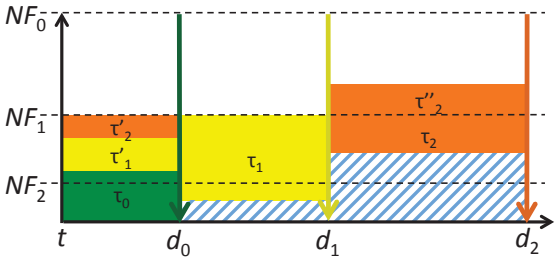

図 5　LBAR の適用例

実行時間の比 AR_i（$0 < AR_i \leq 1$）から計算される．つまり，$u_i^{av} = AR_i \cdot u_i$ である．$[d_{i-1}, d_i]$[*1] において，正規化性能 NF_n で予約できる最大の実行容量として，実行容量閾値 $load\_th_{i,n}$ を以下のように定義する．

$$load\_th_{i,n} = NF_n \cdot (d_i - d_{i-1}) \tag{3}$$

LBAR における初期の予約済みの実行容量 $load_i$ は，

$$load_i = (u_0^{av} + \cdots + u_{i-1}^{av}) \cdot (d_i - d_{i-1}) \tag{4}$$

で求められる．ただし，初期の $load_0$ は 0 である．LBAR では，優先度の高いタスクから順に仕事量を予約していく．仕事量を予約する区間と実行容量を決定するため，$[t, d_0]$ で予約済みの実行容量 $load_0$ について，$load_0$ を実行できる最低の正規化性能 $NF_{n'}$ に注目する．仕事量を予約できる区間の条件は，その区間の予約済みの仕事量が $NF_{n'}$ による実行容量より小さい必要がある．現在時刻から遠いものから順に予約する区間として，$NF_{n'}$ に対応した実行容量を上限に予約する．全区間がこの条件を満たさなくなったら，n' を $n'-1$ に更新して予約の処理を繰り返す．

図 4 は，LBAR によって $load_0$ および NF_{LBAR} を算出する手順を示している．**図 5** に示す具体例を交え，手順を説明する．斜線部分は，初期の予約済みの実行容量を表す．まず，τ_0 の仕事量は，$[t, d_0]$ に予約される．ここで，$[t, d_0]$ の予約済みの仕事量が閾値 $load\_th_{0,2}$ を超えたため，$n' = 1$ となる．次に，τ_1 の仕事量は，まず $[d_0, d_1]$ の NF_1 以下にある実行容量に，残りは図 5 の τ_1' で示す $[t, d_0]$ に予約される．最後に，τ_2 の仕事量は，まず $[d_1, d_2]$ の NF_1 以下に，続けて $[d_0, d_1]$ には NF_1 以下で予約できる実行容量が残っていないため，$[0, d_0]$ の NF_1 以下にある実行容量に予約される．その後，τ_2 の残りの仕事量は，図 5 の τ_2'' で示す $[d_1, d_2]$ に予約される．以上により，予約される平均の仕事量を平準化した正規化性能 NF_{LBAR} を算出できる．

5.3　full look ahead EDF

flaEDF は，仕事量が予約されない部分が生じないように隣接するデッドライン間に仕事量を順に予約し，直近のデッドラインまでの最低の正規化性能を算出する．flaEDF により算出された正規化性能は，最悪時の負荷を用いるためデッドライン制約を保証できる．また，flaEDF では，laEDF において必要以上に高い周波数が算出される問題が解消されている．

図 6 は，flaEDF によって $[t, d_0]$ の実行容量 s および正規化性能を算出する手順を示している．**図 7** に示す具体例を交え，手順を説明する．flaEDF は，laEDF と同様に，優先度の低いタスクから仕事量を予約する．τ_2 の残り最悪実行時間 c_rem_2 による仕事量は，まず，$[d_1, d_2]$ に予約される．ただし，この仕事量は $[d_1, d_2]$ の残り実行容量 $load\_rem = (1 - (u_0 + u_1)) \cdot (d_2 - d_1)$ より大きい．このため，図 7 の τ_2' で示す $[d_0, d_1]$ の実行容量に残りの仕事量を予約する．次に，τ_1 は，$[d_0, d_1]$ の残り実行容量で仕事量が予約できる．最後に，最高優先度の τ_0 について $[t, d_0]$ の実行容量に仕事量を予約する．以上により $[t, d_0]$ に予約された仕事量 s が得られ，デッドライン制約を保証する最低の正規化性能 NF_{flaEDF} が算出される．

5.4　提案手法の適用タイミングと計算の簡単化

提案手法では，LBAR はタスクがリリースされる場合のみ適用するとし，ディスパッチ時には前回の算出結果を用

*1　$d_{-1} = t$ とする．

Input: t, c_rem_i, d_i, u_i, U
Output: $[t, d_0]$ の実行容量 s，NF_{flaEDF}

```
1:  U' ⇐ U
2:  i' ⇐ I − 1
3:  s ⇐ 0
4:  for i = I − 1 to 1 do
5:      c_rem' ⇐ c_rem_i
6:      if i' > i − 1 then
7:          i' ⇐ i − 1
8:          U' ⇐ U' − u_{i'+1}
9:          load_rem ⇐ (1 − U') · (d_{i'+1} − d_{i'})
10:     end if
11:     while i' ≥ 0 do
12:         if load_rem ≥ c_rem' then
13:             load_rem ⇐ load_rem − c_rem'
14:             c_rem' ⇐ 0
15:             break
16:         else
17:             c_rem' ⇐ c_rem' − load_rem
18:         end if
19:         i' ⇐ i' − 1
20:         if i' ≥ 0 then
21:             U' ⇐ U' − u_{i'+1}
22:             load_rem ⇐ (1 − U') · (d_{i'+1} − d_{i'})
23:         end if
24:     end while
25:     s ⇐ s + c_rem'
26: end for
27: s ⇐ s + c_rem_0
28: NF_{flaEDF} ⇐ s/(d_0 − t)
```

図 6　flaEDF による正規化性能の算出

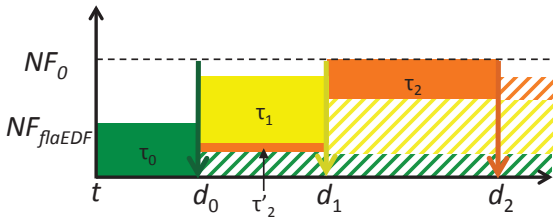

図 7　flaEDF の適用例

いるようにする．LBAR は，平均の仕事量と負荷から平準化した正規化性能を算出するが，これらの値は，タスクのリリースによって隣接デッドライン間の状況が変動しない限り更新の必要が無いためである．また，この場合，$load\_th_{i,n}$ の計算を省略できる．$load\_th_{i,n}$ は，$i \geq 1$ では隣接デッドライン間の時刻の差に依存して計算される．隣接デッドライン間の差は，リリース時に記憶することで，図 4 の 1–5 行目の計算を省くことができる．

次に，flaEDF について考える．flaEDF は，低優先度のタスクから順に仕事量を予約していくため，s に加算されるタスクはある優先度以上のものとなる．ここで，τ_i が実行完了し $s > 0$ である場合は，c_rem_i による仕事量は s に予約されていたことになる．よって，あるタスクの実行完了時には，直前に計算した s から直前に実行されたタスクの以前の c_rem_i を減算するだけでよい．すなわち，

$$s = \max\{0, s - c\_rem_i\} \tag{5}$$

とすればよい．つまり，flaEDF による s の計算はタスクのリリース時のみでよく，次にディスパッチされるタスクがリリースを伴っていない時は，flaEDF を実行せずに $[t, d_0]$ に予約された仕事量 s を計算できる．これにより，タスクのディスパッチ時には，NF_{flaEDF} の算出が簡単化できる．

以上のような処理の簡単化により，ディスパッチ時における提案手法のオーバヘッドを抑えることができる．

6. 評価

6.1 実験環境

提案手法の有効性を評価するため，自作のタスクスケジューリングシミュレータ上で実験を行った．本シミュレータは，与えられたパラメータにより生成されたランダムタスクセットに対して DVFS 手法を適用し，全タスクの周期の最小公倍数であるハイパーピリオドまでの消費エネルギーを算出する．

ランダムタスクセットは，負荷 U および平均と最悪時の実行時間の比 AR をパラメータとして自動生成した．各タスクの起動周期は，2ms 間隔の 2–100ms の一様分布で決定し，ハイパーピリオドが 10s 以下となるタスクを選んだ．各タスクの AR_i は，AR と正規分布から決定した．最悪実行時間および実際の実行時間は，U および AR_i と正規分布から決定した．タスク数 I は 5 個とし，同じ U と AR で生成された 100 個のランダムタスクセットを実行した平均値を評価した．

コアペアテーブルの生成に必要な各コアの IPC，設定できる周波数および消費電力のパラメータは，Cortex-A15 および A7 コアをそれぞれ 4 個もつ Exynos 5422 を搭載する ODROID-XU3 [9] から取得した．パラメータには，ODROID-XU3 上で動作させた Ubuntu 14.04 にて姫野ベンチマーク [10] を実行して計測した値を用いた．

評価対象の DVFS 手法は，(1) 高性能コアのみを動作コアとして使用した laEDF，(2) コアペアテーブルを適用した laEDF（laEDF*），(3) コアペアテーブルを適用した flaEDF（flaEDF），(4) コアペアテーブルを適用した LBAR と laEDF（LBAR+laEDF），および，(5) 提案手法（Proposed DVFS）である．(2)，(3) および (4) は，各手法で正規化性能を算出し，それに対応した動作コアと周波数設定値をコアペアテーブルから決定する．(4) は，LBAR と laEDF で算出した正規化性能のうち，より高いものを選択する．評価では，laEDF と他の手法の有効性を比較するため，(1) の消費エネルギーで結果を正規化した．なお，正規化性能の算出処理および DVFS の適用にかかる時間および消費エネルギーのオーバヘッドは無視した．

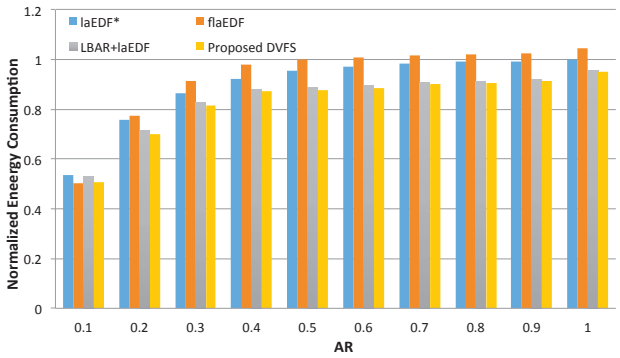

図 8　$U = 0.9$ における評価結果

6.2　評価結果

U には 0.1 から 0.9 まで，AR には 0.1 から 1.0 まで，それぞれ 0.1 刻みを入力パラメータとして，それらの合計 90 通りの組合せを評価した．紙面の都合上，DVFS 手法の効果の特徴が表れている入力パラメータの結果のみを示す．

図 8 は，U を 0.9 とし，AR を変動させた場合の実験結果である．提案手法 Proposed DVFS は，$AR = 0.1$ 以外の場合，実験したタスクスケジューリングのなかで最も高い消費エネルギー削減効果があった．さらに，LBAR+laEDF と比較しても，常に消費エネルギー削減効果が高かった．flaEDF は，$AR = 0.1$ では最も消費エネルギー削減効果が高かったが，$AR > 0.6$ では laEDF より劣る結果となった．

$U = 0.1$ かつ $AR = 0.1$ など，U と AR がともに小さい場合に提案手法の有効性が最大となり，laEDF と比較して最大 59.2%の消費エネルギーを削減できた．laEDF*と比較すると，提案手法は，$U = 0.2$ かつ $AR = 1.0$ の場合に最大 15.0%の消費エネルギーを削減できた．

6.3　考察

提案手法は，他の大部分の入力パラメータに対しても，最も大きい消費エネルギー削減効果があった．従って，提案手法の有効性が示された．

AR と U がともに小さい場合，提案手法および laEDF* の有効性が最も高かった．これは，両者が，コアペアテーブルの適用によって動作コアの適切な切替えを実現し，常に高電力効率コアの正規化性能を算出できたためである．

提案手法は，LBAR+laEDF よりも，消費エネルギー削減効果が高かった．flaEDF は，laEDF の必要以上に周波数が引き上がる問題点が改善されており，laEDF よりも小さい正規化性能を算出できる．このため，提案手法は，LBAR が算出するものを選択する可能性が高く，消費エネルギー削減に寄与したと考えられる．

LBAR および flaEDF は，laEDF よりも複雑な算出手順が必要である．そのため，1 回の算出にかかる時間オーバヘッドは，laEDF よりも大きくなることが予想される．ただし，laEDF は，タスクのディスパッチ時にも実行されるため，その実行回数の予測が困難である．一方，5.4 節で説明した通り，提案手法の本質的な処理はタスクのリリース時のみでよい．従って，提案手法の時間オーバヘッドの予測は，laEDF より容易となる．

以上から，提案手法は，同一命令セットヘテロジニアスマルチコアを採用した組込みリアルタイムシステムのコアペア内のタスクスケジューリングに適しているといえる．

7.　おわりに

本研究では，同一命令セットヘテロジニアスマルチコアにおけるタスクスケジューリング手法を提案した．まず，正規化性能およびコアペアテーブルを定義し，DVFS 手法の適用によってコアペア内の動作コアを切り替える手法を提案した．さらに，各タスクの実行時間の平均値と最悪値の比を用いて正規化性能を算出する LBAR，および，デッドライン制約を保証する最低の正規化性能を既存手法より厳密に算出する flaEDF を提案した．提案手法は，2 種類の方法で算出されたもののうち高い正規化性能で DVFS を適用する．評価の結果，提案手法は，最大で 59.2%の消費エネルギー削減が達成できた．

今後の方針として，DVFS 手法の実行にかかるオーバヘッドを考慮することや，コアペア間におけるタスクスケジューリング手法を検討することが挙げられる．

謝辞　本研究の一部は，JSPS 科研費 24300019 および 26870303 の助成による．

参考文献

[1] Kumar, R., et al.: Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction, *Proc. of MICRO*, pp. 81–92 (2003).
[2] Hu, S. X., et al.: Fundamental of Power-aware Scheduling, *Designing Embedded Processors: A Low Power Perspective*, Springer (2007).
[3] Ishihara, T. and Yasuura, H.: Voltage Scheduling Problem for Dynamically Variable Voltage Processors, *Proc. of ISLPED*, pp. 197–202 (1998).
[4] Pillai, P. and Shin, K. G.: Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems, *Proc. of SOSP*, pp. 89–102 (2001).
[5] Greenhalgh, P.: Big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7, *White Paper* (2011).
[6] NVIDIA: Variable SMP - A Multi-Core CPU Architecture for Low Power and High Performance, *White Paper* (2011).
[7] 高瀬, 他: 排他動作する非均質マルチコアプロセッサとそのリアルタイム OS の実装, 情報処理学会研究報告, Vol. 2014-SLDM-165, No. 15 (2014).
[8] Liu, C. L., et al.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *Journal of the ACM*, Vol. 20, No. 1, pp. 46–61 (1973).
[9] Hardkernel: Odroid XU3 - ODROID [online]. http://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127 (2014).
[10] 理化学研究所情報基盤センター: 姫野ベンチマーク [online]. http://accc.riken.jp/2145.htm (2001).