Vol.2010-EMB-18 No.11
2010/8/10

組込みシステムにおける低消費エネルギー志向の効率的なスラック時間の導出

三輪 遼平[†1] 高瀬 英希[†1,†2] 曾 剛[†3]
冨山 宏之[†4] 高田 広章[†1]

組込みシステムの動的電圧・周波数制御に向いたスラック時間導出手法として，Work-Demand Analysis（WDA）が提案されている．本論文では，次に述べる 2 点について，既存手法 WDA の改良を行う．まず，高優先度タスクからの干渉について，区間を分割して考慮した上でスラック時間の導出を行う．これにより，既存手法と同じ計算量でより最適なスラック時間を導出できる．次に，低優先度タスクへの干渉について，単純化した計算式を提案する．WDA における再帰計算を用いないため，提案手法は計算時間の削減に寄与する．実験の結果，提案手法は，最大で 33 %の消費エネルギーを削減することを確認した．

Low-Energy-Oriented Slack Time Analysis in Embeded Real Time System

Ryohei Miwa ,[†1] Hideki Takase ,[†1,†2] Gang Zeng,[†3]
Hiroyuki Tomiyama [†4] and Hiroaki Takada[†1]

Work-Demand Analysis (WDA) has been proposed in literature for estimation of slack time in an embedded system with dynamic voltage frequency scaling (DVFS) function. While the existing WDA was effective for on-line slack estimation, it had heavy computation and its estimated result was pessimistic. For this reason, we improve the WDA from the above two points in this paper. First, for the calculation of interference time from the high priority tasks, the execution time of task is divided into sections. Thus slack time can be estimated accurately at the same computational cost. Second, for the calculation of interference time from the low priority tasks, a simplified method is proposed to avoid recursion calculation, which results in reduced computation. The experimental results show that the proposed methods can achieve at most 33% energy reduction.

1. はじめに

組込みシステムにおいて，消費エネルギーの量はシステムの価値を決める重要な要素となる．消費エネルギーを削減することによって，製造コストおよび運用コストの低減，システムの信頼性の向上を実現できる．組込みシステムにおける消費エネルギーは，システム全体の中でプロセッサの占める割合が大きい．

組込みシステムのプロセッサの消費エネルギーを削減する手法として，動的電圧・周波数制御（Dynamic Voltage and Frequency Scaling．以降，DVFS）が注目されている．DVFS とは，プロセッサへ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．組込みシステムのプロセッサに一般的に用いられる CMOS 回路の消費エネルギーは，動作周波数の 2 乗に比例すると近似できる[1)]．ゆえに，プロセッサの動作周波数を下げてタスクを実行できれば，システムの消費エネルギーの削減が可能となる．

高いリアルタイム性を要求される組込みシステムにおいては，定められた時刻までにタスクの処理を終えなければならないというデッドライン制約を持つ．このような組込みシステムにおいては，システムのデッドライン制約を保証できる範囲内で DVFS を適用し，タスクの実行時間を伸張させる必要がある．システムのデッドライン制約を保証するためには，動作周波数を切り替えない場合のタスクの残り最悪実行時間から最大の伸張時間を正確に導出することが重要となる．タスクの実行を伸張することができるこの余裕時間を，スラック時間という．スラック時間が楽観的に見積もられると，タスクのデッドライン制約が保証できなくなるおそれがある．いっぽう，スラック時間が悲観的に見積もられると，DVFS による消費エネルギーの削減効果が小さくなる．ゆえに，DVFS の性能は，スラック時間の見積もりの精度に依存するといってよい．

組込みシステムにおけるスラック時間を導出する手法として，2)–6) が提案されている．スラック時間導出手法は，システム動作前に各タスクのスラックを見積もる静的な手法と，

†1 名古屋大学 大学院情報科学研究科
Graduate School of Information Science,Nagoya University
†2 日本学術振興会
The Japan Society for the Promotion of Science
†3 名古屋大学 大学院工学研究科
Graduate School of Engineering, Nagoya University
†4 立命館大学 理工学部
College of Science and Engineering, Ritsumeikan University

システム実行中に見積もる動的な手法とに分類される．本研究では，2) において提案されている Work-Demand Analysis（WDA）というスラック時間導出手法に着目する．WDA は，レートモノトニック・スケジューリング方式のシステムにおいて，動的にタスクスケジューリングを解析する手法である．

本論文では，まず，既存手法 WDA にある 2 つの問題を明らかにする．1 つめの問題は，スラック時間の見積もり対象とするタスクについて，より優先度の高いタスクの実行時間を悲観的に見積もっている点である．2 つめは，より低優先度のタスクの実行時間を見積もるために，再帰計算が必要な点である．そして，本論文において，WDA を起点とした効率的なスラック時間導出手法を提案する．提案手法では，高優先度タスクの実行時間を，より細かく解析区間を分割して見積もる．さらに，対象タスクより優先度の低いタスクに与える実行時間の影響を，単純化した計算式で求める．これらの改善により，WDA より正確にスラック時間を導出することが可能になる．

本論文の構成は，以下の通りである．まず，第 2 章では，提案手法の起点とする WDA の詳細を解説し，その問題点を指摘する．第 3 章では，提案するスラック時間導出手法について述べる．第 4 章では，提案手法の評価と考察を述べる．第 5 章にて関連研究に触れ，最後に，第 6 章にて本論文のまとめと今後の課題を述べる．

2. Work-Demand Analysis

2.1 準　　備

対象とするシステムでは，タスクはレートモノトニック方式でスケジューリングされるとする．n 個のタスクは全て周期的に起動され，$T = \{\tau_1, \tau_2, \ldots, \tau_n\}$ と定義される．タスクの優先度は静的に与えられ，τ の添字が小さいものほど高い優先度をとる．タスク τ_i の j 番目に起動されたインスタンス（起動された個々のタスク）を $\tau_{i,j-1}$ と表記する．各タスクの最初のインスタンスの番号は 0 となる．タスクはすべて独立して動作し，プリエンプションによるタスク切替があるとする．各タスクは，周期 P，相対デッドライン，および，最悪実行時間 ω を持ち，周期と相対デッドラインは等しいとする．各タスクの最初のインスタンスは，時刻 0 においてすべて同時にリリースされる．タスク切り替えにかかる時間は無視できるものとする．DVFS において，動作周波数は最小値 $f_{\min}$ から最大値 $f_{\max}$ までの連続値を選択することが可能であるとする．

2.2 タスクスケジューリングの解析区間

文献 8) によれば，スラック時間導出時刻からシステムのハイパーピリオドまでを解析区間とすれば，各タスクの正確なスラック時間を求めることができる．しかし，タスクスケジューリングの解析区間がハイパーピリオドまでとなると，その計算量は $O(n^2)$ となり実用的ではない．ゆえに，タスクスケジューリング解析に区間を設けることで，計算量を抑える必要がある．

WDA では，各タスクのディスパッチ時にのみスラック時間の導出が行われる．解析区間は，スラック時間導出時刻から各タスクについてその守るべき最初のデッドライン時刻 ud の最大値までとしている．時刻 t における τ_κ の次にあるデッドライン時刻 $ud_\kappa(t)$ は，微少時間を ϵ として以下のようになる．

$$ud_\kappa(t) = \begin{cases} \lceil \frac{t+\epsilon}{P_\kappa} \rceil P_\kappa & if\ \tau_\alpha\ is\ active\ at\ t \\ (\lceil \frac{t+\epsilon}{P_\kappa} \rceil + 1) P_\kappa & otherwise \end{cases} \quad (1)$$

ud は時刻 0 からを基準とした，絶対的な時刻である．

WDA は，対象とするタスクの次に到来するデッドラインまでを解析区間としているため，スラック時間導出にかかる計算量は $O(n)$ となる．

2.3 タスクスケジューリングの解析方法

DVFS によってプロセッサの動作周波数を変更すると，タスクの実行時間は伸縮する．より優先度の高いタスクの実行時間の伸縮がスラック時間導出の対象とするタスクに与える実行時間の影響を，本稿では，高優先度タスクからの干渉と呼ぶ．いっぽう，対象タスクの実行時間の伸縮がより優先度の低いタスクに与える実行時間の影響を，低優先度タスクへの干渉と呼ぶ．

WDA における対象タスクのスラック時間は，対象タスクの残り最悪実行時間 ω^{rem}，高優先度タスクからの干渉 H，および，低優先度タスクへの干渉 L の 3 点を用いて導出される．時刻 t におけるスラック時間 $slack(t)$ は，以下のように求められる．

$$slack(t) = (ud(t) - t) - load(t) \quad (2)$$

$$load(t) = \omega^{rem}(t) + H(t) + L(t) \quad (3)$$

ここで，$load(t)$ は，時刻 t における対象タスクのもつ干渉時間である．

本節では，これらの値の導出方法を説明する．

2.3.1 対象タスクの残り最悪実行時間

ω^{rem} は，次の手順で計算できる．

- タスクの起動時に，ω として初期化する．

- タスクがプリエンプトされて実行を中断した場合は，次のディスパッチ時に，ディスパッチされてからプリエンプトされるまでの時間を ω^{rem} から減算する．
- タスクの実行が終了したら，ω とする．

以上の処理を，タスクの各インスタンスについて繰り返しを行う．

2.3.2 高優先度タスクからの干渉

高優先度タスクからの干渉は，対象タスクについて，より優先度の高いタスク群が対象タスクの ud までに実行する時間のこととなる．WDA では，対象タスクの ud までの H を，解析区間の開始時刻までにリリースされた高優先度タスクが解析区間で実行する時間 H^{past}，および，解析区間の開始時刻から対象タスクの ud までにリリースされる高優先度タスクの実行時間 H^{future} の 2 つに分類して計算している．τ_α の H を H_α とすると，以下のようになる．

$$H_\alpha = H_\alpha^{past} + H_\alpha^{future} \tag{4}$$

時刻 t における $H_\alpha^{past}(t)$ は，ω^{rem} を用いて求めることができ，以下のように表される．

$$H_\alpha^{past}(t) = \sum_{\tau_\kappa \in T_\beta^{ACT}(t)} \omega_\kappa^{rem}(t)$$

where

$$T_\beta^{ACT}(t) = \left\{ \tau_\kappa \mid \kappa < \beta \; and \; \tau_\kappa \in readyQueue(t) \right\} \tag{5}$$

つまり，$H_\alpha^{past}(t)$ は，t において実行可能状態にある高優先度タスクの ω の総和となる．

$H_\alpha^{future}(t)$ は，解析区間内で各タスクがリリースされる回数とそれらのタスクの ω の積の和として計算され，以下のようになる．

$$H_\alpha^{future}(t) = \sum_{i=1}^{\alpha-1} (\lfloor \frac{ud_\alpha(t) - \epsilon}{P_i} \rfloor - \lceil \frac{t+\epsilon}{P_i} \rceil + 1) \cdot \omega_i \tag{6}$$

H は，以下の手順で計算される．

- システムの実行開始時に，各タスクの H は式 (4) によって初期化される．
- タスクがプリエンプトされて実行を中断した場合は，次のディスパッチ時に，ディスパッチされてからプリエンプトされるまでの時間を，プリエンプトされたタスクの H から減算する．
- タスクの実行終了時には，そのタスクの H を式 (4) を用いて計算する．また，実行終了したタスクより優先度の低いタスク群の H から，そのタスクの ω^{rem} だけ減算する．

H の計算量は $O(n)$ となる．

2.3.3 低優先度タスクへの干渉

対象タスクについて，それよりも優先度の低いタスク群のデッドライン制約を保証しつつ，対象タスクの ud までに実行されなければならない時間を，低優先度タスクへの干渉と呼ぶ．対象タスクのスラック時間は，低優先度タスクへの干渉を考慮しつつ求める．

対象タスクのスラック時間 $slack_\alpha(t)$ は，時刻 t における低優先度タスクへの干渉 $L(t)$ と同時に，以下の手順で求められる．

(1) τ_α より低優先度で，かつ，$ud(t) - t$ が最小となるタスク τ_β を求める．

(2) $ud_\alpha(t)$ と $ud_\beta(t)$ を比較する．

(a) $ud_\alpha(t) \geq ud_\beta(t)$ の場合

$H_\beta(t)$ で計算されるタスクは $H_\alpha(t)$ に含まれるため，$H_\alpha(t) < H_\beta(t)$ である．そこで，τ_β のデッドライン制約を保証するために，τ_β のスラック時間を τ_α のスラック時間とする．つまり，式 (3) は，

$$slack_\alpha(t) = (ud_\beta(t) - t) - load_\beta(t) \tag{7}$$

となる．

(b) $ud_\alpha(t) < ud_\beta(t)$ の場合

τ_β のデッドライン制約を保証するため，τ_β が ud_α までに実行される時間を求め，この値を L_α とする．つまり，時刻 t の $L_\alpha(t)$ と $load_\alpha(t)$ は，

$$L_\alpha(t) = \max(0, load_\beta(t) - \omega_\alpha^{rem}(t) - H_\alpha(t) - (ud_\beta(t) - ud_\alpha(t)) \tag{8}$$

$$load_\alpha(t) = \omega_\alpha^{rem}(t) + H_\alpha(t) + L_\alpha(t) \tag{9}$$

となる．

(3) (2) における $load_\beta$ を求めるため，τ_β について (1) および (2) の処理を行う．(1) および (2) の処理を，最低優先度のタスク τ_n まで繰り返す．

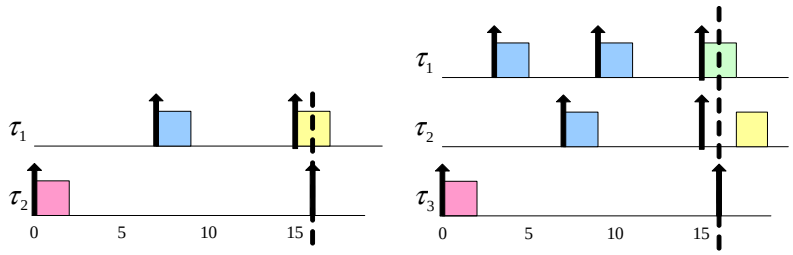
(4) L_n は 0 であるので，$load_n(t)$ が導出できる．τ_α まで計算を戻っていくことで，$L_\alpha(t)$，$load_\alpha(t)$，および，$slack_\alpha(t)$ の値が求まっていく．

以上のように，WDA では再帰計算によって L を求める．L の導出にかかる計算オーバヘッドはタスク数を n として $O(n)$ である．

2.4 既存手法の問題点

本節では，WDA にある 2 つの問題点を指摘する．

Vol.2010-EMB-18 No.11
2010/8/10



(a) ケース 1　　(b) ケース 2

図 1　既存手法 WDA で H を悲観的に計算する例
Fig. 1　Example of pessimistically calculating H in existing technique

まず，式 (6) において，H^{future} が悲観的に計算されているという点である．高優先度タスクからの干渉は，対象タスクより高優先度のタスク群が最悪実行時間で実行されるという仮定のもとで計算される．しかし，解析区間内で最後にリリースされるタスクのインスタンスは，最悪実行時間の分全てが対象タスクに干渉するわけではない．このため，H の計算値が悲観的になることがある．

H が悲観的に計算されるケースは，以下のように分類される．

ケース 1　解析区間内でリリースされる高優先度タスクの最後のインスタンスが，対象タスクの ud をまたいで実行される場合．このとき，対象タスクの ud を超えて実行した分も含めた H^{future} を導出してしまう．**図 1(a)** の例では，τ_1 の最後のインスタンスは，τ_2 の ud_2 をまたいで実行している．ud_2 以降に実行される τ_1 の時間も余分に含めて，H_2^{future} が悲観的に計算される．

ケース 2　複数のより優先度の高いタスクのインスタンスが，対象タスクの ud をまたいで実行される場合．このとき，対象タスクの ud を超えて実行される複数の高優先度タスクの実行時間を含めた H^{future} が導出されてしまう．**図 1(b)** の例では，τ_1 および τ_2 の最後のインスタンスが同時に ud_3 をまたいで実行されている．

われわれの提案する手法は，これらのケースでも適切なスラック時間を導出できる．

WDA の 2 つめの問題として，低優先度タスクへの干渉を再帰計算により求めている点が挙げられる．再帰計算を用いると，タスク数が多くなるにつれてシステム動作中の計算時間が大きくなるおそれがある．ゆえに，再帰計算に依らないスラック時間導出手法を確立する必要がある．

3. 効率的なスラック時間導出手法

提案手法では，2.4 節で述べた WDA の 2 つの問題点を改善する．3.1 節で H の計算における悲観的な計算部分の改善点，3.2 節でスラック時間の導出方法における再帰的な計算部分の改善点を説明する．

3.1 悲観的な計算部分の改善

われわれは，H^{future} をより詳細に計算することで，WDA の悲観さを低減することを実現する．提案手法では，H^{future} を $H^{future'}$ および H^{last} の 2 つに分割し，高優先度タスクからの干渉 H を以下のように求める．

$$H = H^{past} + H^{future'} + H^{last} \tag{10}$$

ここで，H^{past} は，WDA の式 (5) と同様に計算できる．

式 (10) における H^{last} とは，時刻 t から対象タスクの ud までの区間のうちで，最後にリリースされる高優先度タスク群のインスタンスの実行時間の合計である．対象タスクより優先度の高いタスクを 1 つのみ扱える手法を 3.1.1 節で，複数の高優先度タスクを扱える手法を 3.1.2 節で提案する．

$H^{future'}$ は，解析区間内でリリースされるそれぞれの高優先度タスクの最悪実行時間の総和から，各タスクの最後にリリースされるインスタンスの実行時間を除いたものである．つまり，式 (10) における $H^{future'}$ は，WDA における H^{future} から前述の H^{last} を除外したものである．τ_α の時刻 t における $H^{future'}(t)$ は，以下のように計算される．

$$H_\alpha^{future'}(t) = \sum_{i=1}^{\alpha-1}(\lfloor\frac{ud_\alpha(t)-\epsilon}{P_i}\rfloor - \lceil\frac{t+\epsilon}{P_i}\rceil)\cdot\omega_i \tag{11}$$

$H^{future'}(t)$ は，対象タスクより優先度の高い全てのタスクを考慮して計算されるため，着目するタスクの個数は限定しない．

3.1.1 Effective-WDA1

1 つめの提案手法である Effective-WDA1 では，解析区間内でリリースされる高優先度タスクの最後のインスタンスが，対象タスクの ud をまたいで実行される場合（図 1(a)）に対応する．Effective-WDA1 における H^{last} は，以下の式であらわされる．

Vol.2010-EMB-18 No.11
2010/8/10

$$H_{\alpha}^{last}(t) = \sum_{i=1}^{\alpha-1} \min(\omega_i, ud_{\alpha}(t) - \lfloor \frac{ud_{\alpha}(t) - \epsilon}{P_i} \rfloor \cdot P_i) \quad (12)$$

つまり，対象タスク τ_{α} より優先度の高いタスク τ_i について，解析区間内の最後にリリースされるインスタンスの実行が ud_{α} をまたがない場合は ω_i，またぐ場合は ud_{α} と τ_i の最後のリリース時刻の差を全ての高優先度タスクについて求め，その総和が H_{α}^{last} となる．

3.1.2 Effective-WDA2

2 つ目の提案手法である Effective-WDA2 は，優先度の高い複数のタスクのインスタンスが，対象タスクの ud をまたいで実行される場合に対応できる．h_i^{last} は，ある高優先度タスクの解析区間内で最後にリリースされるインスタンスが，対象タスクに干渉を与えるかをあらわす 0-1 変数である．$H_{\alpha}^{last}(t)$ は，次に示すように，τ_i のインスタンスが対象タスクの ud をまたいで実行される場合とまたがない場合とに分けて考慮される．

$$H_{\alpha}^{last}(t) = (ud_{\alpha}(t) - release_{\min}) \cdot g^{last}(t) + \sum_{i=1}^{\alpha-1} (1 - h_i^{last}(t)) \cdot \omega_i \quad (13)$$

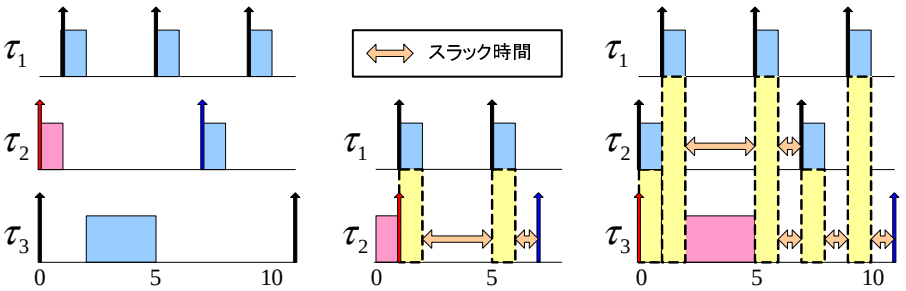
$$h_i^{last} = \begin{cases} 1 & if\ (\lfloor \frac{ud_{\alpha} - \epsilon}{P_{\alpha}} \rfloor \cdot P_i + \omega_i) > ud_{\alpha} \\ 0 & otherwise \end{cases} \quad (14)$$

$$g^{last}(t) = \begin{cases} 1 & if\ \sum_{i=1}^{\alpha-1} h_i^{last} \geq 1 \\ 0 & otherwise \end{cases} \quad (15)$$

ここで，$release_{min}$ は，ud_{α} をまたいで実行される高優先度タスク群のインスタンスのリリース時刻の最小値をあらわす．

τ_i のインスタンスのいずれか 1 つが対象タスクの ud をまたいで実行される場合は，そのインスタンスのリリース時刻の中から最小値 $release_{\min}$ を求める．このとき，対象タスクの $ud(t)$ と $release_{\min}$ の差を H_{α}^{last} に加算する．この加算は，ud_{α} をまたぐタスクの個数に依らないことに注意されたい．Effective-WDA2 は，対象タスクが複数の高優先度タスクから干渉を受ける場合に対応できている．

Effective-WDA1 および Effective-WDA2 の計算量は，ともに $O(n)$ となる．つまり，提案する 2 つの手法は，WDA と同様の計算量で，高優先度タスクからの干渉をより最適に導出することができる．



(a) タスクスケジューリング例　(b) $slack_2^{withoutL}$ の導出　(c) $slack_3^{withoutL}$ の導出

図 2 提案手法のスラック時間導出方法
Fig. 2 Slack Time Analysis of proposal technique

3.2 再帰的な計算部分の解消

われわれは，さらに，WDA における低優先度タスクへの干渉の計算を単純化する．

提案手法における低優先度タスクへの干渉の計算には，$slack^{withoutL}$ を利用する．$slack^{withoutL}$ とは，低優先度タスクへの干渉を考慮せずに導出する対象タスクの仮のスラック時間のことである．対象タスクの実行時間の伸張によりデッドラインミスが起きる可能性のあるタスクは，対象タスクとより優先度の低いタスク群のみである．つまり，全てのタスクのデッドライン制約を保証するためには，対象タスクと低優先度タスク群の $slack^{withoutL}$ の最小値を対象タスクのスラック時間とすればよい．つまり，提案手法において低優先度タスクへの干渉を考慮したスラック時間は，以下のようになる．

$$slack_{\alpha}(t) = \min_{\alpha \leq i \leq n} (slack_i^{withoutL}(t)) \quad (16)$$

$$slack_i^{withoutL}(t) = (ud_i(t) - t) - H_i(t) - \omega_i^{rem}(t) \quad (17)$$

例えば，**図 2(a)** において，時刻 0 における τ_2 のスラック時間を導出するとする．この場合，対象タスク τ_2 と低優先度タスク τ_3 の $slack^{withoutL}$ を求める．まず，$slack_2^{withoutL}$ を求める．τ_2 の時刻 0 から ud_{τ_2} までの低優先度タスクを無視したタスクスケジューリングは，**図 2(b)** のようになる．H_2 は τ_1 の時刻 0 から ud_{τ_2} までの実行時間となるため，2 となる．τ_2 の残り最悪実行時間は 1 なので，$slack_2^{withoutL}$ は $7 - 2 - 1$ で 4 となる．次に，$slack_3^{withoutL}$ を求める．τ_3 の時刻 0 から ud_{τ_3} までの低優先度タスクを無視したタスクス

ケジューリングは，**図 2(c)** のようになる．H_3 は τ_1 と τ_2 の，時刻 0 から ud_{τ_3} の間の実行時間となるため，$3+2$ で 5 なる．τ_3 の残り最悪実行時間は 3 より，$slack_3^{withoutL}$ は，$11-5-3$ で 3 となる．ゆえに，$slack^{withoutL}$ の最小値である 3 が，対象タスク τ_2 のスラック時間となる．

われわれの提案手法は，WDA と同精度の L が導出される．さらに，再帰的な処理を含まないため，L の導出にかかる計算時間の削減が期待できる．

4. 評価実験

4.1 評価環境

提案手法の有効性を評価した．評価には，各タスクのディスパッチ時にスラック時間を導出し，そのスラック時間に応じて DVFS を適用した際の消費エネルギーを用いた．実験は，自作のタスクスケジューリング・シミュレータを用いた．比較対象として，以下の手法による消費エネルギーをそれぞれ評価した．

WDA 文献 2) において提案された手法である Work-Demand Analysis によってスラック時間を導出する

Effective-WDA1 高優先度タスクからの干渉を 3.1.1 節で提案した手法によって，低優先度タスクへの干渉を 3.2 節で提案した手法によってスラック時間を導出する

Effective-WDA2 高優先度タスクからの干渉を 3.1.2 節で提案した手法によって，低優先度タスクへの干渉を 3.2 節で提案した手法によってスラック時間を導出する

動作周波数設定の方針としては，各タスクのディスパッチ時にのみタスクセットの消費エネルギー $E_{dynamic}$ は，文献 1)，2) を参考として，以下の式を用いて求めた．

$$E_{dynamic} \propto \sum_i f_i^2 \cdot NC_i \tag{18}$$

ここで，f_i はプログラム実行中に設定された動作周波数，N_c は f_i の設定値で実行したサイクル数をあらわす．なお，静的なエネルギーであるリークエネルギーは考慮せず，実行すべきタスクがない時の消費エネルギーは 0 とした．システムの動作開始時からハイパーピリオド（全てのタスクの周期の最小公倍数）までを測定区間とし，その消費エネルギーを計算した．

実験内容および評価に用いたタスクセットは，文献 2) を参考にした．タスクセットは，以下の条件でランダムに作成する．

- タスクセットの実行時間は，最悪実行時間に対する実行時間の割合をパラメータとして変化させる．また，各インスタンスの実行時間は一定である．
- タスクの周期は，10 ns から 100 ns の間でランダムに設定される．
- 各タスクの最悪実行時間は，1 ns からタスクの周期の間でランダムに設定される．

表 1 実験 1 で用いたタスクセット
Table 1 Task set for Experiment1

タスク	周期（ns）	ω（ns）	タスク	周期（ns）	ω（ns）
τ_1	10	2.28	τ_4	50	3.86
τ_2	10	0.73	τ_5	65	8.73
τ_3	25	7.08	τ_6	85	3.09

提案手法の有効性を評価するため，2 種類の実験を行った．実験 1 では，最悪実行時間に対する実行時間の割合を 0.1，0.2，0.3，0.4，0.5，0.6，0.7，0.8，および，0.9 にそれぞれ設定した．実験 1 に用いたタスクセットを，**表 1** に示す．実験 2 では，タスク数を 2，4，6，8，および，10 個とした．それぞれのタスク数について，ランダムに 10 種類のタスクセットを評価し，各手法における消費エネルギーの平均値を算出した．実験 2 では，CPU 使用率を 0.9 に，各タスクの最悪実行時間に対する実行時間の割合を 0.5 と設定した．

4.2 実験結果

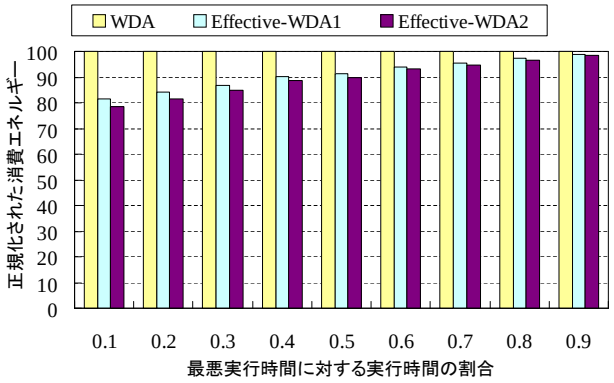
実験 1 および実験 2 の結果を，**図 3(a)** および**図 3(b)** に示す．図中の消費エネルギーは，WDA における結果を 100 として正規化した値を示している．

4.3 考察

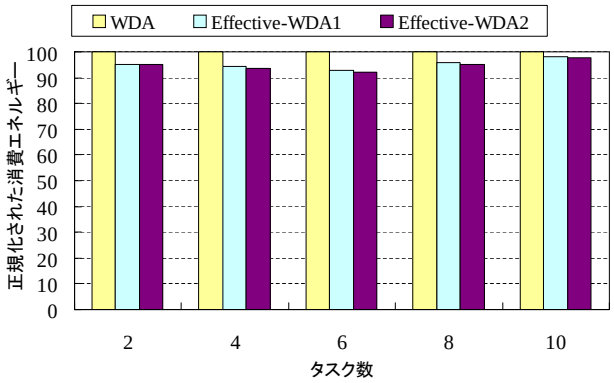
図 3(a) および図 3(b) をみると，本論文の提案手法である Effective-WDA1 および Effective-WDA2 は，全ての状況において，既存手法 WDA よりも DVFS の消費エネルギー削減効果が大きくなっていることがわかる．DVFS によって消費エネルギーが削減できたということは，より多くのスラック時間を適切に導出できたことを示している．つまり，WDA の問題点である高優先度タスクからの干渉の計算結果の悲観さが，提案手法によって低減できている．

2 つの提案手法を比較すると，Effective-WDA2 のほうが Effective-WDA1 よりも DVFS による消費エネルギー削減の効果が大きいことがわかる．3.1 節で解説したとおり，Effective-WDA2 は，対象タスクより優先度の高いタスクが複数存在する場合に対応している．図 1(b) に示したような，対象タスクの ud をまたいで複数の高優先度タスクが実行される場合でも，Effective-WDA2 は高優先度タスクからの干渉を適切に導出できている．

実験 2 では，既存手法よりも消費エネルギーを削減できないタスクセットが存在した．こ



(a) 実験 1



(b) 実験 2

図 3 実験結果（消費エネルギー）
Fig. 3 Experimental results (energy consumption)

れは，評価に用いるために作成したタスクセットの性質で説明できる．これらのタスクセットは，対象タスクの ud をまたいで実行される高優先度タスクのインスタンスが存在しなかったと考えられる．2.4 節および 3.1 節で論じたように，提案手法は，高優先度タスクからの干渉が悲観的に計算されている場合に優位性が生まれる．しかし，タスクの周期の差が大きい場合には，既存手法でも高優先度タスクからの干渉を正確に計算できる場合がある．よって，タスク数が多く，かつ，タスク間の周期の差が小さいタスクセットであるほど，提案手法の有効性は高くなると考えられる．

提案手法は，3.2 節で解説したように，低優先度タスクへの干渉を単純化して求める．これにより，スラック時間導出にかかる計算時間の削減が期待される．しかし，本研究では，この計算時間の比較評価はできなかった．リアルタイム OS 上に提案手法を実装しての計算時間の評価は，今後の課題とする．

5. 関 連 研 究

これまでに，タスクのスラック時間を導出し，これを DVFS に適用してシステムの消費エネルギーを削減する研究がいくつか行われている．

文献 3) は，システム開始前にスラック時間を導出する maximum constant speed を提案している．スケジューリング判定式を用いてデッドライン制約を保証したうえでの最大の周波数によってそれぞれのタスクを実行する．文献 4)，5) では，優先度ベーススラックスティーリングという手法が提案されている．優先度ベーススラックスティーリングでは，あるタスクが最悪実行時間よりも早く実行を終了した場合，そのスラック時間は次に実行するタスクに割り当てられる．タスクのデッドライン制約を保証するため，スラック時間を割り当てるタスクは，より優先度の低いもののみに限られる．文献 3) では，Stretching-to-NTA という手法が提案されている．この手法では，タスクの実行開始時に，他のあるタスクが次にリリースされる時刻までにスラック時間が存在すれば，DVFS により動作周波数を下げるようにする．Pillai らは，Stretching-to-NTA を拡張し，自タスクのみならず実行可能状態にある他のタスクにもスラック時間を分配できる手法を提案している[6]．

文献 7)，9) では，maximum constant speed，優先度ベーススラックスティーリング，および，本論文で着目した WDA の有効性を比較した研究が行われている．Kim らが評価した手法のうちでは，動的優先度のシステムでは優先度ベーススラックスティーリングが，固定優先度のシステムでは WDA が，最も有効であることが示されている．

6. お わ り に

本研究では，レートモノトニック方式の組込みシステムを対象とした効率的なスラック時間導出手法を提案した．提案手法は，既存研究において提案された Work-Demand Analysis (WDA) を起点としている．本論文では，まず，WDA にある 2 つの問題点を指摘した．ま

ず，高優先度タスクからの干渉が悲観的に計算されていた点が挙げられる．これは，高優先度タスクの解析区間をより分割することで導出されるスラック時間の悲観さを低減した．2つめの問題点は，低優先度タスクへの干渉に再帰計算を用いている点である．これは，対象タスク以下の優先度のタスクについて，仮のスラック時間の最小値のみをことで，計算の単純化を実現した．提案手法を DVFS に適用して評価したところ，最大で 33 %の消費エネルギーの削減を確認した．

今後の方針としては，提案したスラック時間導出手法をリアルタイム OS に実装しての評価や，非周期タスクへの対応などが挙げられる．

謝辞　本研究の一部は，独立行政法人科学技術振興事業団（JST）戦略的創造研究推進事業（CREST）「情報システムの超低消費電力化を目指した技術革新と統合化技術」の支援による．

参 考 文 献

1) Henkel, J. and Parameswaran, S.(eds.): *"Designing Embedded Processors: A Low Power Perspective"*, chapter9, Springer (2007).
2) Kim, W., Kim, J. and Min, S.L.: "Dynamic voltage scaling algorithm for fixed-priority real-time systems using work-demand analysis", *Proceedings of the International Symposium on Low Power Electronics and Design*, pp.396–401 (2003).
3) Shin, Y., Choi, K. and Sakurai, T.: "Power Optimization of Real-Time Embedded Systems on Variable Speed Processors", *Proceedings of the International Conference on Computer-Aided Design*, pp.365–68 (2000).
4) Aydin, H., Melhem, R., Mosse, D. and Alvarez., P.M.: "Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems.", *Proceedings of IEEE Real-Time Systems Symposium*, pp.95–105 (2001).
5) Kim, W., Kim, J. and Min, S.L.: "A Dynamic Voltage Scaling Algorithm for Dynamic-Priority Hard Real-Time Systems Using Slack Time Analysis", *Proceedings of Design, Automation and Test in Europe*, pp.788–794 (2002).
6) Pillai, P. and Shin, K.G.: "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems", *presented at 18th ACM Symposium on Operating Systems Principles*, pp.89–102 (2001).
7) Kim, W., Shin, D., J.Jeon, J.K. and Min, S.L.: "Performance Evaluation of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems", *Journal of Low Power Electronics*, Vol.1, pp.1–11 (2005).
8) Lehoczky, J.P. and Thuel, S.R.: "An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems.", *Proceedings of the IEEE Real-Time Systems Symposium*, pp.110–123 (1992).
9) Kim, W., Shin, D., J.Jeon, J.K. and Min, S.L.: "Performance Comparison of Dynamic Voltage Scaling Algorithms for Hard Real-Time Systems", *Proceedings of Real-Time and Embedded Technology and Applications Symposium*, pp.219–228 (2002).