Vol.2020-SLDM-192 No.28
2020/11/18

Responsive Linkを用いた分散リアルタイムシステムにおけるRT-DVFS手法

鈴木宏海† 山﨑信行†

† 慶應義塾大学 大学院理工学研究科
E-mail: †{suzuki,yamasaki}@ny.ics.keio.ac.jp

あらまし 近年の組込みリアルタイムシステムにおいては，システムの大規模化に伴って複数のCPUをネットワークで接続した分散リアルタイムシステムが多く存在する．これらのシステムの消費電力を低減する手法として，タスクをノード間でマイグレーションしながら実行するRT-DVFS手法が数多く提案されているが，これらのアルゴリズムはタスクがCPU間でマイグレーションされる際の時間的なオーバーヘッドを考慮しておらず，ネットワーク帯域が有限である実システムでのリアルタイム性の保証が困難である．本研究では，リアルタイム通信用に開発された通信規格であるResponsive Linkを用いてタスクマイグレーションのスケジューリングを行い，リアルタイム性を保証しながら動的な負荷分散を行う手法を提案し，提案手法をResponsive Linkを搭載したSoCであるRMT Processsor上のソフトウェアに実装する．実機を用いて測定した消費電力をもとにしたシミュレーションによって提案手法がシステム全体の消費電力を削減できることを示す．また，RTLシミュレーションによってソフトウェアがマイグレーションのスケジューリングに要する時間的オーバーヘッドを評価する．

キーワード 組込みリアルタイムシステム，分散リアルタイムシステム, RT-DVFS，Responsive Link, RMT Processor

Real-Time Voltage and Frequency Scheme using Responsive Link for Distributed Real-Time Systems

Hiromi SUZUKI† and Nobuyuki YAMASAKI†

† Graduate School of Science and Technology Keio University
E-mail: †{suzuki,yamasaki}@ny.ics.keio.ac.jp

Abstract In the field of Real-Time embedded systems, distributed Real-Time systems which connect several processors in a network becomes major due to large scale of systems. In order to reduce power consumption and guarantee a met deadline, several RT-DVFS methods that allow tasks to migrate between processors are proposed. However, these algorithms don't consider time overheads when a task is migrated between processors. Therefore, it is difficult to guarantee a feasible schedule in a real system. This paper proposes a migration scheduling method for RT-DVFS using the Responsive Link designed for real-time communications. The method is implemented on the RMT processor which integrates the Responsive Link. We evaluate the energy saved by our aproach on simulation, based on parameters obtained from the SiP and also evaluate software scheduling overhead on RTL simulation.

Key words embedded real-time，distributed real-time, RT-DVFS，Responsive Link, RMT Processor

1. まえがき

機器等に組込まれる組込みシステムにおいては，システムの消費電力は可能な限り少ないことが要求される．これは組込みシステムの多くがバッテリー駆動であり，過剰に電力を消費するシステムは稼働時間の低下やバッテリーコストの増大を招くためである．システム内部の消費電力削減手法の1つとしてDynamic Voltage and Frequency Scaling (DVFS) が挙げられる．これはCPUの消費電力がその動作電圧と周波数に依存することを利用し，それらの値をソフトウェアによって変化させることで，消費電力を抑える手法である．DVFSは消費電力とパフォーマンスの間にトレードオフを発生させるため，システム内のタスクに実行完了までの時間制約が付与されているリアルタイムシ

ステムにおいてはシステムのリアルタイム性を保ったままシステムの負荷に応じて動作電圧と周波数を決定する Real-Time DVFS (RT-DVFS) が用いられる．シングル CPU に対する RT-DVFS 手法としては Cycle-Conserving (CC) [1] や Look-Ahead (LA) [1]，Dynamic Reclaiming Algorithm (DRA) [2] などが広く知られている．

また近年では組込みリアルタイムシステムの大規模化に伴い，ネットワークで接続された複数のノードに処理を分散させる分散リアルタイムシステムが広く採用されている．分散リアルタイムシステムにおいてもパーティションスケジューリングを用いたシングル CPU での RT-DVFS 手法が有効であるが，より一層の消費電力の削減にはタスクマイグレーションを用いてタスクを各ノードに適切に分散させる必要がある．これらの研究も盛んに行われているが [3] [4] [5]，これらのアルゴリズムは全てのタスクが任意の時点においてノード間を時間的オーバーヘッド無しにマイグレーション可能であるという仮定を敷いている．一般にネットワークの帯域幅はメモリ帯域幅に対して小さいためタスクマイグレーションに要する時間は同一ノード内でのコンテキストスイッチなどに比べて非常に大きく，加えて実システムにおいてはノード間の位置関係やネットワーク負荷によってオーバーヘッドが大きく変化する．そのため，このオーバヘッドを無視することはアルゴリズムと実システムの乖離によるデッドラインミスの増加やスケジューラビリティの低下を齎す．

本研究では，リアルタイム通信用に開発された通信規格である Responsive Link [6] を用いることでタスクマイグレーションの時間的オーバーヘッドを予測し，リアルタイム性を保証しながら動的な負荷分散を行う手法を提案する．本研究ではまた，提案手法を Responsive Link を搭載した SoC である Responsive Multithreaded Processor (RMTP) [7] [8] 上のソフトウェアに実装するための設計を提案する．

2. 背　　景

2.1 Responsive Link

Responsive Link [6] は ISO/IEC で国際標準化されている分散リアルタイムシステム向けの通信規格である．Responsive Link では通信路をイベントリンクとデータリンクの 2 種類に分離しており，それぞれ制御信号などの低レイテンシの求められるハードリアルタイム通信と画像や音声といった高スループットの求められるソフトリアルタイムに適するように設計されている．イベントリンクはパケットサイズを抑えることで転送レイテンシを削減し，データリンクではパケットサイズを大きく取ることで実効スループットを向上させている．

各通信リンクは point-to-point の全 2 重通信であり，ルーティングテーブルによって任意のトポロジーを設定できる．Responsive Link は通信パケットに対して 256 諧調で優先度を付与することで，各ノードの送信バッファ内で高優先度パケットが低優先度パケットを追い越すことが可能であり，通信のスケジューリングを可能としている．追い越されたパケットが送信バッファから溢れた場合，内臓の SDRAM コントローラによって SDRAM へ退避され，バッファに空きができた場合に自動で復帰する．また，伝送路エラーによる再送は通信のレスポンスの時間予測性を致命的に低下させるため，再送を防ぐためにラインコードレベル，バイトレベル，ブロックレベルでそれぞれ ECC を搭載し，高いエラー耐性を備えている．

2.2 Responsive Multithreaded Processor

Responsive Multithreaded Processor (RMTP) [7] [8] は組込みリアルタイムシステム，分散組込みリアルタイム向けに開発された SoC であり，分散リアルタイムシステム用途に Responsive Link を内蔵している．RMTP では高い面積・電力効率を達成するために SMT アーキテクチャを採用しており，スレッド毎の実行速度の予測性を高めるため，各スレッドに対してハードウェアで優先度を付加することでスレッド間の命令発行優先度を調停している．また，割込みに対して 1 クロックでスレッドを起動状態に遷移できる割込み起床機構を備え，I/O 処理に対して高い応答性を持つ．

3. 提 案 手 法

3.1 システムモデル

本研究ではノード間が Responsive Link によって point-to-point で接続され，通信が複数のノードを経由して送信されるネットワーク持つ分散リアルタイムシステムを想定する．分散システムは M 個のノード $node_i$ とヒープを利用しない N 個の互いに独立した周期リアルタイムタスク τ_j から構成される．各タスクは周期ごとにジョブがリリースし実行される．タスク τ_j はデッドラインと等しい周期 $T_j (0 < T_j)$ と最悪実行時間 $C_j (0 < C_j < T_j)$ が与えられる．タスクの CPU 使用率 u_j は $\frac{C_j}{T_j}$ と定義され，ノードの CPU 使用率 U_i はそのノード内に存在する全てのタスクの CPU 使用率の合計である．また，CC アルゴリズムに基づくタスクの実 CPU 使用率を l_j，ノードの実 CPU 使用率を L_i と定義する．

3.2 マイグレーションが可能なタスクの条件

[4] では partitioned CC-EDF をベースにノードの負荷に応じて実行途中のタスクを 1 周期分のみマイグレーションする re-partitioning による RT-DVFS 手法を提案している．この手法はベースのスケジューリングにタスクマイグレーションを行う必要のないパーティションスケジューリングを用いているため，輻輳等でマイグレーションが困難になった場合でもリアルタイム性を維持できるという利点がある．そのため，本研究においても re-partitioning 手法を採用する．本論文ではタスクがシステム開始時に割当てられたノードをホームノードと定義し，マイグレーションが発生しない限りタスクはホームノード上で実行される．

タスクをマイグレーションして実行するためには，タスクの命令とデータをマイグレーション先のノードに転送しなければならない．タスクが持つデータの内，命令と静的領域に配置されたデータはその転送量を容易に予測可能である一方で，任意の時刻におけるタスクのスタック領域の使用量を予測することは困難であり，マイグレーションレスポンスの時間予測性を低下させる大きな要因である．そこで本研究ではリアルタイムシ

```
int staticval; //静的領域に展開される 変数

void periodic_task()
{
   int stackval ; //スタック領域に展開される 変数
   while (true) {
      stackval = ... //  変数がループ毎に独立
      staticval = staticval + 1; //変数がループを跨いで更新
      ... //  周期的な処理
      wait_period (); //  次周期まで待機
   }
}

voidtask_start()
{
   ... //初期化処理
   periodic_task();
}
```

図 1　周期タスクの記述例

ステムの周期タスクの性質に着目して，マイグレーションレスポンスの時間予測性を向上させる．

リアルタイムシステムの周期タスクは一般に，図 1 のようにループによって記述され，ループの末尾でジョブの実行が完了する．周期タスク内部で使用される変数のうち，周期を跨いで値を保持する変数は静的領域に確保する．一方でスタック領域には周期毎に独立した値を持つ変数のみを配置することで，ジョブの実行完了時にはスタックを破棄することができる．この性質を利用し，本研究ではマイグレーション可能なタスクをマイグレーション開始時刻において既にその周期の実行が終了しているタスクと定義する．これにより，タスクマイグレーションに必要な転送量を命令を含めたタスクの静的データ部分のみに限定し転送量の予測性を大幅に高めることができる．本研究では τ_j の実行前に行われる転送を onward 転送と定義し，輻輳が発生しない場合の最悪のレスポンス時間を $wcrt_j^{on}$ と表す．同様に逆方向の転送を homeward 転送と定義し，レスポンス時間を $wcrt_j^{home}$ で表す．

マイグレーションされたタスクは，次の周期のみをマイグレーション先のノード $node_{dst}$ で実行する．周期毎に更新されるデータがある場合，次々回のジョブの起床までに $node_{dst}$ で更新されたデータがホームノード上に書き戻されていなければならない．図 2 はある時刻 t_0 でタスクマイグレーションが開始されたタスクの $node_{dst}$ での実行例である．タスクはジョブの起床後かつ onward 転送が完了した時刻 r_j' に実行可能となり，次の周期の起床までに homeward 転送を完了するため d_j' までに実行が終了していなければならない．タスク τ_j の $node_{dst}$ での CPU 使用率 u_j' は式 1 のように与えられる．

$$\begin{aligned} u_j' &= \frac{C_j}{slack} \\ slack &= d_j' - r_j' \\ &= T_j - R_{home}(\tau_j, d_j) - \max\left(0, R_{on}(\tau_j, t_0) - (r_j - t_0)\right) \end{aligned} \tag{1}$$

ここで，$R_{on}(\tau_j, t_0)$，$R_{home}(\tau_j, d_j)$ はその時刻からまたはその時刻までに τ_j の onward または homeward 転送を行うときの通信の輻輳を加味したレスポンス時間である．

タスクが $node_{dst}$ でスケジュール可能であるかは式 2 に示す d_j' における実 CPU 使用率 $L'_{dst,d_j'}$ を元に判定する．$L'_{dst,d_j'} \leq 1$ の時スケジュール可能である．

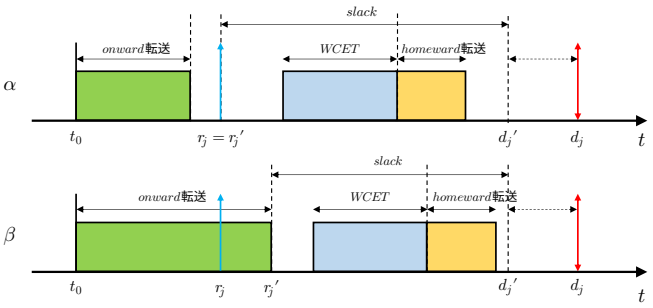

図 2　マイグレーションされたタスクの実行例

$$L'_{dst,d_j'} = u_j' + \sum_{\tau_k \in node_{dst}} \begin{cases} u_k & (\tau_k \text{ release until } d_j') \\ l_k & (\tau_k \text{ release after } d_j') \end{cases} \tag{2}$$

R_{on} と R_{home} はタスクのマイグレーション実行の可否解析に影響を与えるため，ある時刻で計算されたレスポンス時間がそれ以降の時刻において増大することは避けなければならない．そこで，先に開始された onward 転送により高いパケット優先度を与えることで，後から開始されたマイグレーションによって一度求めたレスポンス時間が増大することを防止する．一方で homeward 転送はタスクの実行終了後に開始されるため，リアルタイム性を高めるためにタスクが最悪実行時間より早期に終了した場合にはスケジュールより前倒して開始される．前倒された homeward 転送による onward 転送のプリエンプションを避けるため，homeward 転送は onward 転送より低い優先度が与えられる．タスクマイグレーションのそれぞれの転送における優先度を式 3 に示した．$P_{on}(\tau_j)$，$P_{home}(\tau_j)$ はそれぞれ τ_j をマイグレーションする際の優先度であり，n はタスクマイグレーションがこれまでに発生した回数を表す．P_{max} は付与できる優先度の最大値である．提案手法はタスクマイグレーションが発生するたびに優先度を徐々に下げていくため，連続してスケジュールできるタスクマイグレーションの最大数は $P_{max}/2$ 個に制限される．

$$\begin{aligned} P_{on}(\tau_j) &= P_{max} - n \\ P_{home}(\tau_j) &= P_{max}/2 - n \end{aligned} \tag{3}$$

アルゴリズム 1 にタスク τ_j が $node_{src}$ から $node_{dst}$ に転送される場合の onward 転送のレスポンス時間を予測する疑似コードを示した．各経路において，$wcrt_j^{on}$ 分の転送が可能な時間を順に探索する．ある経路においてその時間に他のマイグレーションがスケジュールされていない場合，onward 転送は他のマイグレーションにブロックされることなく行われる．一方で他のマイグレーションがスケジュールされていた場合，タスク τ_j の onward 転送は先に始まったマイグレーションによってブロックされるためレスポンス時間が増大する．スケジュールが homeward 転送と重複した場合，onward 転送はブロックされず，重複しているマイグレーションの homeward 転送のレスポンス時間が増大してしまうため，そのタスクはマイグレーションすることができない．

アルゴリズム 2 はアルゴリズム 1 同様に homeward 転送のレスポンス時間を予測する疑似コードである．homeward 転送は τ_j のジョブのデッドラインまでに完了しなければならないため，

Algorithm 1 Predict τ_j Onward Response Time

Require: args τ_j, $node_{src}$, $node_{dst}$
Ensure: $R_{on}(\tau_j, now)$
1: **if** $T_j < wcrt_j^{on}$ **then**
2: unschedulable
3: **end if**
4: $routes \leftarrow$ list of path links from $node_{src}$ to $node_{dst}$
5: $s \leftarrow now$, $e \leftarrow s + wcrt_j^{on}$
6: **for** $link_i$ in $routes$ **do**
7: **while** $true$ **do**
8: **if** schedule is free at $[s, e]$ on $link_i$ **then**
9: reserve schedule if needed
10: break
11: **else**
12: **if** overlap with homeward transfer **then**
13: flush reserved schedule
14: **return** unschedulable
15: **else**
16: $s \leftarrow$ end of overlap, $e \leftarrow s + wcrt_j^{on}$
17: **end if**
18: **end if**
19: **end while**
20: **end for**
21: commit reserved schedule
22: **return** e

Algorithm 2 Predict τ_j Homeward Response Time

Require: args τ_j, $node_{src}$, $node_{dst}$
Ensure: $R_{home}(\tau_j, d_j)$
1: $routes \leftarrow$ list of path links from $node_{dst}$ to $nodesrc$
2: $e \leftarrow d_j$, $s \leftarrow e - wcrt_j^{home}$
3: **for** $link_{ri}$ in $reverse(routes)$ **do**
4: **while** $true$ **do**
5: **if** schedule is free at $[s, e]$ on $link_{ri}$ **then**
6: reserve schedule if needed
7: break
8: **else**
9: $e \leftarrow$ start of overlap, $s \leftarrow e - wcrt_j^{home}$
10: **end if**
11: **end while**
12: **end for**
13: commit reserved schedule
14: **return** $e - s$

パケットの宛先である $node_{src}$ 側から時間を遡ってスケジュールを探索する．homeward 転送が他の転送と重複した場合，現時点でスケジュールされている全ての転送は τ_j の homeward 転送より高い優先度を持つため，homeward 転送はブロックされる．そのため，τ_j のデッドラインまでに転送を完了するために転送を前倒しでスケジュールする．

これらのアルゴリズムにより，輻輳したリンクを避けたタスクマイグレーションが可能であり，輻輳によるマイグレーションタスクのデッドラインミスを防止することができる．

リアルタイム性を維持できる場合でもタスクマイグレーションによって消費電力を削減できることが期待されない限りマイグレーションを行うべきではない．$node_{src}$ 及び $node_{dst}$ において，時刻 r'_j における実 CPU 使用率 L'_{src,r'_j} 及び，L'_{dst,r'_j} を予測し，式 4 を満たすタスクをマイグレーションによる消費電力の削減が可能なタスクであると判断する．

マイグレーションのスケジューリングはジョブの起床/完了のスケジューリングイベント毎に行われ，ジョブの起床時にはそのジョブが起床したノードを $node_{src}$ としてノードに割当てられているタスクからマイグレーション可能なタスクを探索する．同じタスクが複数のノードに対してマイグレーション可能である場合には最もホップ数の少ないノードを選択するものとする．一方でジョブの完了時にはそのジョブが完了したノードを $node_{dst}$ としてタスクを探索する．本研究では既にマイグレーションされてホームノード以外に割当てられているタスクのマイグレーションを禁止し，1 つのスケジューリングイベントに対してマイグレーションされるタスクの最大数を 1 つとする．対象となるタスクが複数ある場合，最も起床時刻が近いタスクをマイグレーションする．対象となるノードが複数ある場合，最もホップ数の少ないノードへマイグレーションする．

$$|L'_{src,r'_j} - L'_{dst,r'_j} - u_j - u'_j| < L'_{src,r'_j} - L'_{dst,r'_j} \qquad (4)$$

4. 設計と実装

本章では提案手法の RMTP 上のソフトウェアへの設計と実装を述べる．提案したスケジューラーは応答性を高めるため RMTP のハードウェアスレッドを 1 本専有し割込み起床機構による処理を行う．Responsive Link は 2.1 で述べたようにレイテンシを抑えたイベントリンクとスループットを高めたデータリンクを備える．本研究ではスケジューリングイベントとマイグレーションイベントの通知をイベントリンクを用いて行い，ノード間のタスクマイグレーションをデータリンクを用いて行う．また，マイグレーションスケジューラーの時間分解能はタスクのスケジューラーと同一とする．また転送量を削減するため onward 転送では送信元のデータは保持し，homeward 転送では静的領域データのみを転送する．

4.1 パケットフォーマット

各ノードでスケジューリングイベントが発生した際，マイグレーションスケジューラーを起動するためにノードはイベントの発生を通知する．マイグレーションがスケジュールされた場合には $node_{src}$ に対してマイグレーションイベントの通知を行う．通知を受けたノードは $node_{dst}$ に対して onward 転送を開始する．実行完了時には $node_{dst}$ から $node_{src}$ に向かって homeward 転送を行う．この一連の流れを実行するためには (1) スケジュールイベントの通知，(2) マイグレーションイベントの通知，(3)onward 転送，(4)homeward 転送の 4 つの通信を行う必要がある．先に述べたように本研究では (1)(2) をイベントリンクで，(3)(4) をデータリンクを用いて通信を行う．

各通信における Responsive Link のパケットフォーマットを図 3 に示す．イベントリンクは 1 パケット 16Byte であり，ヘッダとトレイラを除いた 8Byte がペイロードである．データリ

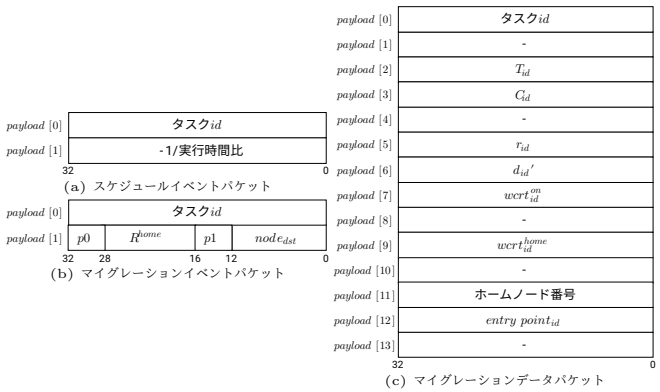

図 3　タスクマイグレーションのパケット構成

ンクは 1 パケット 64Byte でありペイロードは 56Byte である．これらは受信時に割込みを発生させる割込みパケットであり割込起床機能によってスレッドを起動する．(1) では *payload*[0] にイベントの対象となるタスク id が，*payload*[1] に起床時には-1 が，完了時には実実行時間/最悪実行時間比が格納される．実行時間比は必ず正の値を取るため，この 2 つは MSB によって区別可能である．マイグレーションスケジューラーには事前にシステム全体のタスクセットのパラメータを与え，この情報を用いて各ノードの CPU 使用率を更新する．

(2) ではマイグレーションを行うタスク id と $node_{dst}$，優先度を伝える必要がある．必要な情報を 1 パケットに収めるために R_{home} のフィールドを圧縮している．また *payload*[1] のビットフィールドを Responsive Link のヘッダ部と揃えることで，パケットを受けたノードは R_{home} を自身のノード番号に書き換えるだけでヘッダを構成可能である．

(3)(4) において図 3(c) はそれぞれの転送の開始を示すパケットであり，このパケットに続いてタスクのバイナリデータを送信する．(3)(4) は同一のパケットフォーマットを用いて行われ，ノードは自身のノード番号とパケット内のホームノード番号を比較して onward 転送と homeward 転送を区別する．フォーマットはスケジューラーのタスク情報構造体を一部流用するためマイグレーションに必要のないデータを格納している空間は省略した．$node_{dst}$ でタスクを実行するにはスタックの破棄が可能な実行開始 PC を知る必要がある．それを示すのが entry point フィールドであり，初期化処理の最後にカーネルコールによって登録することで図 1 のループ開始位置を指し示す．それぞれのパケットはトレイラのポート番号によって区別する．

以上の設計を元に，RMTP のソフトウェア上に実装を行った．新たに追加した API のうち大きなものとその役割を表 1 に示す．

5. 評　　価

本研究で提案するアルゴリズム及びソフトウェアの有効性を評価するため，3. のアルゴリズムを実装したシミュレーターによって消費電力を評価する．また，4. を実装したソフトウェアに対して cadence 社の xcelium を用いた RTL シミュレーションによって実装のオーバーヘッドを評価した．

表 1　実装した API

function name	function
is_migratable	あるタスクが特定のノードへマイグレーション可能であるかを判定する
rl_migrate_ctrl	ノード使用率の更新とマイグレーションのスケジューリングを行う
rl_sta_onward	$node_{dst}$ へ onward 転送を開始する
rl_sta_homeward	ホームノードへ homeward 転送を開始する
rl_rsv_onward	onward 転送の受信を開始する
rl_rsv_homeward	homeward 転送の受信を開始する

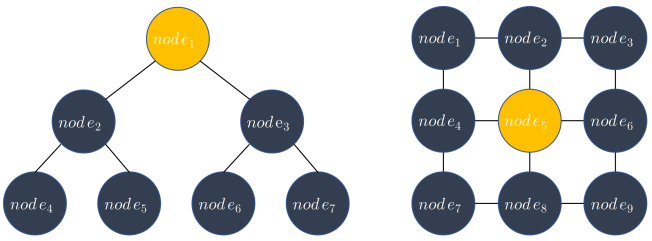

図 4　想定するネットワークトポロジー

表 2　DRMTP 動作周波数倍率と動作電圧及び消費電力

	RMT PU 動作周波数倍率			
	1.00	0.50	0.33	0.25
動作電圧	1.07 V	0.86 V	0.82 V	0.78 V
RMT PU 消費電力	0.92 W	0.45 W	0.36 W	0.26 W
Responsive Link 消費電力	0.082 W	0.048 W	0.030 W	0.024 W

対象となるシステムとして図 4 に示すような tree と mesh の 2 種類のネットワークトポロジーを想定する．mesh では XY ルーティングを想定する．色の異なるノードはマスターノードであり，マイグレーションのスケジューリングを行う．

消費電力の評価はアルゴリズムを実装したシミュレーターによって行う．Dependable RMTP (DRMTP) の消費電力を測定し，シミュレーションにおける RMT PU の消費電力，及び Responsive Link の設定可能な動作周波数倍率と動作電圧における消費電力を表 2 のように設定した．本実装ではタスクのスケジューリングと実行を行うスレッドとマイグレーションのスケジューリングを行うスレッドが並列実行されるため，それを模して RMT PU は Mibench を 2 スレッドで同時実行した際の消費電力を測定した．Responsive Link はラインコードを 8B10B，DPLL の分解能を 8 分周，バイトレベル及びブロックレベルの ECC を使用しないとして設定した．また，RMTP の Responsive Link は RMT PU と電源を共有しているため，動作周波数を RMTP の最低動作周波数と揃えて測定した．なおこの設定におけるデータリンクの実効スループットは 2.1825Mbps である．

5.1　消費電力に関するシミュレーション

アルゴリズムに対してランダムに生成したタスクセットを与え，消費電力を評価した．タスクの周期は{20, 50, 100, 200}ms から選択され，CPU 利用率が 5% から 50% の範囲になるよう最悪実行時間を設定した．最悪 onward レスポンス時間は最大 20ms とし，homeward レスポンス時間は 0 から onward レスポンス時間と等しい範囲で設定した．全てのパラメータは一様なランダム値によって生成された．CPU の平均使用率は 10% から 90% まで 10% 毎に 100 個のタスクセットを生成した．タスクセットは Worst-Fit Decreasing (WFD)，もしくは Best-Fit Decreasing (BFD) によってパーティショニングを行い，ホームノードを設定した．全てのタスクのジョブの実実行時間 (AET) は最悪実行時間の{25, 50, 75}% とする。それぞれのノードは CC-EDF が適用されている．動作電圧は動作周

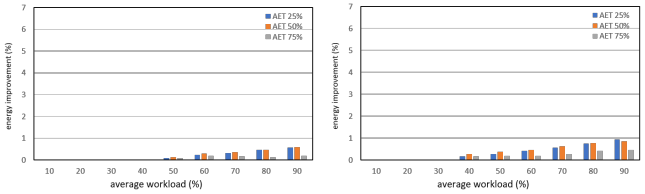

図 5　WFD における消費電力のシミュレーション結果（左:tree，右:mesh）

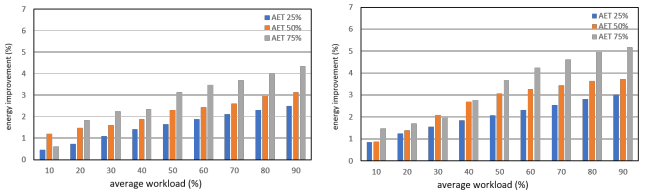

図 6　BFD における消費電力のシミュレーション結果（左:tree，右:mesh）

波数に対して最適な値に即座に変更されるものとした．スケジューリングが可能だったタスクセットを対象に RMT PU とデータリンクの電力消費量についてハイパーピリオドの 5 倍までシミュレーションを行い，消費電力を評価した．

各平均 CPU 使用率におけるシミュレーション結果を図 5，6 に示した．各平均 CPU 使用率において，提案手法を用いず CC-EDF のみを行った場合の消費電力から提案手法の適用により削減された消費電力の割合を示している．partitioned CC-EDF でスケジューリング可能だったタスクセットの全てにおいて提案手法はスケジューリング可能であった．WFD でパーティショニングした場合に比べ，BFD でパーティショニングした場合の方が消費電力の削減割合が高く，最大で 5% 消費電力を削減できている．これは BFD により発生した CPU 使用率の偏りに対して多くのタスクマイグレーションが行われた為と考えられる．また，tree に対して僅かに mesh の消費電力の削減割合が多いことから提案手法はネットワークの輻輳に対して有効に機能していると考えられる．

5.2　オーバーヘッドに関する RTL シミュレーション

RMTP 上に実装した提案手法の演算オーバヘッドの評価を以下のように行った．5.1 で消費電力の評価に用いたタスクセットの中で，tree トポロジーにおいてタスクマイグレーションの回数が多かったタスクセットを選び，同様のパラメータを持ったタスクセットを与えて RTL シミュレーションを行った．計測区間はスケジュールイベントパケットを受け取ったマスターがマイグレーションスケジューラーを起動してからタスクマイグレーションの実行を判断し，実行する場合にはマイグレーションイベントパケットを送信するまでとし，スケジューリングに要したクロックサイクル数を測定した．測定結果を表 3 に示す．スケジューリングに要したクロックサイクル数はイベントごとに大きくばらついていることが示された．これはノードに割当てられたタスク個数やノードの数，既にスケジューリングされているマイグレーションの数などで探索対象となるタスクやノードの数，1 タスクのマイグレーション可能性判定にかかるクロックサイクル数が変化するためと考えられる．

マイグレーションスケジューラーの演算オーダーはノード数 M，タスク数 N，平均ホップ数 H に対して $O(M \times N \times H)$ である．測定内での最悪値は CPU の速度設定によってはスケジューラーの時間分解能に近くなるため，最悪値に近い探索が連続した場合にはシステムのパフォーマンスに影響を与える可能性がある．

表 3　マイグレーションスケジューラーの実行クロックサイクル数

	最小値	最大値	中央値	平均値	標準偏差
クロックサイクル数	1349	13771	2985.5	3905	300

6. ま と め

本研究ではリアルタイム通信用に設計された Responsive Link を用いて，タスクマイグレーションにかかる時間的オーバーヘッドを考慮しながらノードへのタスクの割当てを動的に re-partitioning することでリアルタイム性を維持しながら消費電力を低減するアルゴリズムを提案し，RMTP 上のソフトウェアとして設計実装した．

評価では，提案手法がリアルタイム性を維持しながらタスクマイグレーションを行い，消費電力を最大 5% 削減できることがシミュレーションにより示された．また，RMTP 上のソフトウェアへの設計実装によって，提案手法が実時間において実行可能であることが RTL シミュレーションにより示された．

今後の研究課題としては，ネットワークが高負荷にある状態での動作の安定化，演算オーバヘッドの削減，及び実システムによる消費電力の評価などが挙げられる．

謝辞　本研究は国立研究開発法人 科学技術振興機構 A-STEP (AS2815003R) の助成を受けたものであることを記し，謝意を表す．

文　献

[1] P.Pillai and K.G.Shin, "Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems," Proceedings of the 18th ACM Symposium on Operating Systems Principles, pp.89-102, 2001.
[2] H.Aydin, R.Melhem, D.Mosse, and P.Mejia-Alvarez. "Power-aware scheduling for periodic real-time tasks," IEEE Trans. Comput., 53:584–600, 2004.
[3] M.K.Bhatti, C.Belleudy and M.Auguin, "An inter-task real time DVFS scheme for multiprocessor embedded systems," 2010 Conference on Design and Architectures for Signal and Image Processing (DASIP), Edinburgh, 2010, pp. 136-143.
[4] E.Seo, J.Jeong, S.Park and J.Lee, "Energy Efficient Scheduling of Real-Time Tasks on Multicore Processors," IEEE Transactions on Parallel and Distributed Systems, vol. 19, no. 11, pp. 1540-1552, Nov. 2008
[5] L.Zheng, "A Task Migration Constrained Energy-Efficient Scheduling Algorithm for Multiprocessor Real-time Systems," 2007 International Conference on Wireless Communications, Networking and Mobile Computing, Shanghai, 2007, pp. 3055-3058
[6] N.Yamasaki, "Responsive Link for Distributed Real-Time Processing," Innovative architecture for future generation high-performance processors and systems (iwia 2007), Maui, HI, 2007, pp. 20-29.
[7] K.Suito and R.Ueda and K.Fujii and T.Kogo and H.Matsutani and N.Yamasaki, "The Dependable Responsive Multithreaded Processor for Distributed Real-Time Systems," IEEE Micro, vol.32, pp.52-61, 2012.
[8] S. Nakabeppu et al., "Space Responsive Multithreaded Processor (SRMTP) for Spacecraft Control," 2020 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS), Kokubunji, Japan, 2020, pp. 1-3.