情報処理学会論文誌 Vol. 52 No. 12 3729–3744 (Dec. 2011)

実行トレースを用いた組込みシステムにおけるタスク内DVFSのためのチェックポイント抽出

立松知紘[†1,*1] 高瀬英希[†1,†2] 曾剛[†1]
川島裕崇[†1] 冨山宏之[†3] 高田広章[†1]

近年，組込みシステムにおける消費エネルギーの削減は，重要な課題となっている．タスク内 DVFS によって消費エネルギーを効果的に削減するためには，チェックポイントと呼ばれる，プログラム内部で動作周波数を切り替える場所が重要となる．さらに，消費エネルギー削減効果を向上させるためには，適切な数のチェックポイントが挿入されるのが望ましい．そこで，本研究では，実行トレースを用いたチェックポイントの抽出手法を提案する．本手法は，チェックポイント候補を抽出するための実行トレースマイニングと，より効果の高いチェックポイントを選定するチェックポイント選定法で構成される．実行トレースマイニングでは，実行時における条件分岐命令での振舞いと残り実行サイクル数を抽出して集約することでチェックポイント候補を探索する．チェックポイント選定法では，チェックポイント候補を消費エネルギー削減効果の観点で順位付けを行い，挿入したい個数のチェックポイントがプログラムに挿入される．評価実験では，MediaBench の jpeg エンコーダとデコーダのベンチマークに対して，提案手法を 2 種類の DVFS 戦略に適用し，その有効性を確認した．

Execution Trace Based Checkpoint Extraction for Intra-task DVFS in Embedded Systems

Tomohiro Tatematsu,[†1,*1] Hideki Takase,[†1,†2]
Gang Zeng,[†1] Hirotaka Kawashima,[†1]
Hiroyuki Tomiyama[†3] and Hiroaki Takada[†1]

In recent years, it has been a crucial issue to reduce the energy consumption of embedded systems. For intra-task DVFS, it is important to decide the location of checkpoints in a program, at which the frequency of processor could be changed to reduce the energy consumption. Furthermore, since insertion of checkpoints causes both time and energy overhead, it is expected that only limited and effective checkpoints should be inserted. In this work, execution trace based checkpoint extraction for intra-task DVFS is proposed, which consists of two steps. In execution trace mining, the behaviors and remaining execution cycles of conditional branch instructions are extracted and merged from a large number of traces, and this information is used to help search checkpoint candidates. In checkpoint extraction, checkpoint candidates are ranked in order of descending efficiency for energy reduction, and only the expected number of top-ranked checkpoints is inserted into the program. The proposed technique was evaluated by using the jpeg encoder and decoder benchmarks, and its effectiveness has been confirmed.

1. はじめに

組込みシステムにおける消費エネルギーの削減は，システムの製造コストや運用コストの抑制，環境および生活の保全など，様々な恩恵を与えることができる．特に，バッテリを電源とする組込みシステムでは連続駆動時間の向上にも貢献する．以上のように，消費エネルギーは製品価値を左右する大きな要素となるため，消費エネルギー削減の要求は今日では大きなものとなっている．そこで，これらの要求に対応するべく，組込みシステムの消費エネルギーの最適化を目指す研究がこれまでに数多く行われている．

組込みシステムの消費エネルギー削減技術の 1 つに，動的電圧周波数制御（DVFS: Dynamic Voltage and Frequency Scaling）がある．DVFS とは，プログラム実行時にプロセッサの動作周波数および供給電圧を切り替える技術のことである．電圧は動作周波数と比例関係にあり，消費エネルギーは動作周波数の 2 乗と実行サイクル数の積に比例すると近似できる[1)]．ゆえに，DVFS によって動作周波数を降下させることによって，プロセッサの消費エネルギーを削減することができる．しかし，動作周波数の降下は実行時間の増大に直結する．リアルタイム性が要求される組込みシステムでは，デッドラインと呼ばれる特定の時刻までに処理を終えなければならないという時間制約がある．このため，この時間制約を保

†1 名古屋大学
Nagoya University
†2 日本学術振興会
Japan Society for the Promotion of Science
†3 立命館大学
Ritsumeikan University
∗1 現在，西日本旅客鉄道株式会社
Presently with West Japan Railway Company

証するよう動作周波数を適切に設定する必要がある．

タスク内 DVFS とは，タスクの実行中のある地点でプロセッサの動作周波数を切り替える技術である．タスク内 DVFS によって，効率良く消費エネルギーを削減するためには，動作周波数を切り替える地点が重要である．本研究では，この地点をチェックポイント[2] と呼ぶ．プログラムがチェックポイントに到達すると，最適な動作周波数の計算が行われ，必要であれば動作周波数の切替えが行われる．

タスク内 DVFS の戦略を取り上げた研究として，文献 3)–6) がある．これらの研究では，プログラムの制御フローグラフ（CFG: Control Flow Graph）から，各 DVFS 戦略で用いる残り実行サイクル数の見積りが変動する箇所を求め，チェックポイントを挿入する．しかし，CFG を解析する手法では，CFG 上では存在するがどのような入力データを与えてもプログラム実行時には実際に通過することのないパスも考慮されることがある．また，大規模なプログラムでは，その CFG を取得して解析することは容易ではない．さらに，チェックポイントの処理において，動作周波数および供給電圧を切り替えるべきと判断された場合には，DVFS の処理にともなう，実行時間と消費エネルギーのオーバヘッドがかかる．文献 7) によれば，商用のプロセッサでは，動作周波数および供給電圧の切替えにおおよそ 200 μs から 500 μs ほどのオーバヘッドがかかるといわれている．チェックポイントを多く挿入すると，DVFS により動作周波数および供給電圧を切り替える機会が頻発し．このオーバヘッドの影響が無視できないものになりうる．そのため，特に消費エネルギー削減効果の高いチェックポイントを選定する必要がある．

本研究では，シングルタスクで構成されるプログラムにおいて，タスク内 DVFS に有効なチェックポイントを，実行トレースマイニングを用いて抽出する手法を提案する．そして，タスク内 DVFS に提案手法を適用することによって，デッドラインの制約を保証しつつ，プロセッサの平均消費エネルギーの最小化を目指す．提案手法は，大きく分けて，実行トレースマイニングによってチェックポイント候補を探索する処理と，チェックポイント候補から特に消費エネルギー削減効果のあるものを選定する処理で構成される．

実行トレースマイニングとは，プログラムの大量の実行トレースを解析し，プログラムからタスク内 DVFS に有益な情報を機械的に導き出す処理を指す．実行トレースマイニングは，実行トレースのみを入力として適用可能な解析手法である．高信頼ソフトウェアの開発時には，プログラムの様々なパスを通過するようなテストケースや，最悪実行時間を解析するためのテストケースを与えることが多い[8]．また，今日では，組込みシステムの信頼性を確保するため，網羅性の高いテストケースの生成を支援することを目的とした商用ツールも販売されている[9]．このように生成されたテストケースをプログラムに与えることで，網羅性の高い実行トレースを取得して，プログラムの様々な特徴を抽出することができる．これまでにも，プログラム内で実行頻度の高いパスを特定するなどの目的のために，実行トレースを用いた解析方法が提案されている[10),11]．本研究における提案手法は，ソフトウェア開発時に生成されるテストケースを流用し，そこから得られる実行トレースの集合を入力として適用できる．実行トレースマイニングでは，実行時における条件分岐命令での振舞いと，条件分岐命令後の残り実行サイクル数を抽出して集約する．条件分岐命令に着目するのは，その振舞いに応じて，今後の処理における残り実行サイクル数の見積りが変動する可能性が高いためである．そして，この集約した情報を用いて，対象となるプログラムからチェックポイント候補の探索を行う．

提案手法では，さらに，チェックポイント候補の中から，特に消費エネルギー削減に効果のあるものを選定する．チェックポイントの選定は，オーバヘッドを考慮したグリーディ法に基づいて行われる．チェックポイントの残り実行サイクル数の見積りや消費エネルギー削減効果は，採用する DVFS 戦略によって変わる．本研究では，DVFS 戦略として 2 種類の戦略に着目する．1 つめは，チェックポイント以降の最悪実行サイクル数を採用した DVFS 戦略であり，2 つめは，チェックポイント以降で最も実行される確率の高いパスの実行サイクル数に着目した DVFS 戦略である．提案手法により選定されたチェックポイントが対象となるプログラムに挿入され，デッドラインを満たしながらタスク内 DVFS の運用によって組込みシステムの消費エネルギー削減が達成される．

本研究の貢献は次のとおりである．

- 実行トレースに着目した手法であるため，タスクの実行時の振舞いがより正確に解析できる．実行トレースの集合は，ソフトウェア開発時のテストや品質保証のために生成されるテストケースを用いることで容易に取得することができ，プログラムの CFG を解析する必要はない．また，実行トレースの集合のみを扱うため，CFG 解析できない大規模なプログラムにおいても，提案手法は適用可能である．
- チェックポイントを挿入することによって発生するオーバヘッドを考慮したうえで，チェックポイントの選定を行う．消費エネルギー削減への寄与量によってチェックポイントを評価し，その有効性によって順位付けする．チェックポイントの評価は，採用する DVFS 戦略に応じて行われる．
- プログラムには，指定された個数のチェックポイントが挿入される．順位付けされたチェックポイントの上位の集合を抽出することで，消費エネルギー削減の有効性が高い

チェックポイントのみが挿入される．

本論文の構成は以下のとおりである．まず，2 章で既存のタスク内 DVFS に関する研究について触れる．3 章では提案手法の全体像について説明する．4 章で，チェックポイント候補を探索するための実行トレースマイニングについて解説し，5 章で採用する DVFS 戦略，および探索されたチェックポイント候補から，チェックポイントを選出する処理について述べる．6 章では，提案手法の有効性を検証するための評価実験について述べる．最後に，7 章でまとめと今後の展望について触れる．

2. 関 連 研 究

DVFS を用いて消費エネルギーの削減を目指した研究は，これまで数多く行われている．本章では，その中でタスク内 DVFS に関する研究について述べる．

文献 3)–6) では，対象となるプログラムの CFG を利用して DVFS を行う手法が提案されている．これらの手法では，一部の基本ブロック間にチェックポイントを置き，このポイントで動的に，もしくは静的に動作周波数が定められる．

文献 3) では，残り実行サイクル数の見積りとして，プログラム終了までの最悪実行サイクル数を採用する手法が提案されている．この手法では，まず，静的に残り最悪実行サイクル数が減少する基本ブロック間にチェックポイントが挿入される．そして，実行時には，チェックポイントに到達するたびに，新しく動作周波数を計算し，切り替えながら実行を続ける．

文献 3) の手法では，通過する確率が低く，実行サイクル数が多いパスが含まれている場合に，大きな消費エネルギー削減効果は期待できない．そこで文献 4) では，チェックポイント以降で，最も通る確率の高いパスの実行サイクル数を利用した手法が提案されている．チェックポイントは，最も通る確率の高いパスから外れる基本ブロック間に挿入される．

文献 4) の手法は，通過する確率がある程度高いパスが複数存在するときに，効果を発揮できない可能性がある．そこで，Kumar らは文献 5) において，複数の通過する確率の高いパスを考慮した仮想パスを用いた手法を提案している．

文献 3)–5) の手法では，必ずしもプログラムの平均消費エネルギーを最適化できるとはいえない．そのため，文献 6) では，平均消費エネルギーが最小となる実行サイクル数を持つ仮想的な基本ブロックを用いる，最適実行パスを用いた DVFS 手法が提案されている．最適実行パスとは，DVFS において動作周波数を決定する際，平均消費エネルギー最小に，かつ，デッドライン制約を満たせられるような実行サイクルを持つパスを指す．この実行サイクル数は，後続の基本ブロックの実行サイクル数と，通過確率によって定められる．

これらの手法は，CFG 上の基本ブロックで，各 DVFS 戦略における残り実行サイクル数を算出し，それが変動する箇所にチェックポイントを挿入している．しかし，実際に通過しうるパスを把握することができないため，適切なチェックポイントを挿入できない可能性がある．さらに，これらの研究では，対象となるプログラムの CFG が取得できることが前提となっている．だが，文献 8) でも述べられているとおり，大規模なプログラムからその CFG を解析することは容易でなく，また，CFG 解析の商用ツールにおいては対象のソースコードが構造化プログラミングなどの規約に準拠するといった制約を受ける．つまり，CFG 解析を用いてチェックポイントを抽出する手法は，大規模なプログラムに対しては現実的ではないと考えられる．我々の手法は，開発時に生成されるテストケースおよび実行トレースのみを入力として扱うため，ソースコードや CFG の解析は不要となる．また，これらの既存手法では，解析により選定されたチェックポイントは，その有効性の多寡によらずすべてプログラムに挿入される．

CFG とは別の観点から DVFS を行う手法も過去に提案されている．文献 12)，13) では，プログラムの実行時間の確率分布から DVFS を行う手法が提案されている．プログラムは入力データによって，実行時間が変動することが一般的である．これらの手法では，事前にプログラムの実行開始時は動作周波数を小さくしておき，時間が経過するにつれて動作周波数を増加させる手法を提案している．動作周波数の増加度合いは，プログラムの実行時間の統計情報から決定する．しかし，割り当てる動作周波数の決定には，膨大な計算量がかかるため，近似アルゴリズムを用いて，最適な動作周波数の割当てを多項式時間で計算できるようにしている．文献 12) ではシングルタスク環境を対象としているが，文献 13) では，文献 12) の手法をマルチタスク環境へ拡張している．

3. 全　体　像

本章では，提案手法全体の概要について述べる．

本研究の対象は，シングルタスク環境のシステムである．すなわち，システムの処理単位（タスク）が 1 つのみ存在するシステムである．タスクの実行パスとプログラムに与えられる入力データには依存関係があり，残り実行サイクル数が異なる実行パスを含むプログラムを対象としている．タスクの実行パスは，主に条件分岐命令の振舞いによって変わりうる．このようなシステムにおいて，タスクの実行中のある地点でプロセッサの動作周波数を切り替える技術であるタスク内 DVFS を適用する．

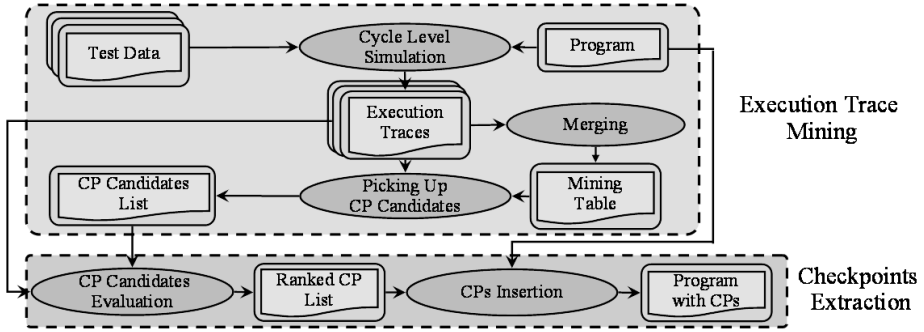

図 1　提案手法のワークフロー
Fig. 1　The workflow of the proposed approach.

提案手法のワークフローを**図 1** に示す．提案手法は，チェックポイント候補の探索のための実行トレースマイニングと，チェックポイント候補から採用する DVFS 戦略に特に効果のあるチェックポイントの選定の 2 つの段階に大別できる．

本研究における提案手法は，ソフトウェアの開発時において，その信頼性の確保や最悪実行時間の解析のために生成されるテストケースを入力として扱う．実行トレースマイニングは，これらのテストケースを流用し，そこから得られる実行トレースの集合のみを扱うことで適用できる．テストケースによる実行トレースのパスの網羅性は，リアルタイム性の保証と関係がある．たとえば，入力の実行トレースが網羅していないパスをプログラムが通過し，かつ，そのパスがテストケースにより網羅したパスより実行時間が長い場合は，デッドラインミスが発生する可能性がある．本提案手法は，最悪実行時間を保証したうえでの平均消費エネルギーの最小化を目指す．そのため，入力として与えられるテストケースおよび実行トレース集合においても，最悪実行時間を解析できるテストケースが含まれることが前提となる．さらに，より精確にプログラムの実行時間を見積もるためには，より多くのテストケースが与えられることが望ましい．

実行トレースマイニングでは，網羅性の高い大量の実行トレース群を解析し，最悪実行パスによる DVFS 戦略，および最頻実行パスによる DVFS 戦略に有効と考えられるチェックポイント候補の箇所が探索される．本研究では，残り実行サイクル数の見積りが変動する地点を探索するために，残り最悪実行サイクル数が減少する条件分岐命令に着目する．実行トレースマイニングは，実行トレース群からマイニングテーブルを作成するフェーズと，取得したマイニングテーブルと実行トレース群からチェックポイント候補を探索するフェーズの 2 フェーズから構成される．前者のフェーズでは，まず，サイクルレベルシミュレーションによって取得した実行トレース群から，対象となるプログラムにおける条件分岐命令に関する情報を取得する．具体的には，条件分岐命令のアドレス，同一の条件分岐命令の実行順，振舞い，および，残り実行サイクル数の一連の情報を取得する，そして，取得した実行トレースから得られた情報が，マイニングテーブルへ集約される．後者のフェーズでは，生成されたマイニングテーブルと，はじめに取得した実行トレースを用いて，対象となるプログラムに挿入されるチェックポイントの候補が探索される．これらの操作の詳細は，4 章で述べる．

チェックポイントの選定は，実行トレースマイニングの操作の後に行われる．具体的には，実行トレースマイニングによって生成されたチェックポイント候補リストから，特に効果のあるチェックポイント候補を選定する．これは，数多くのチェックポイント候補と入力データを与えて DVFS を試行することで，チェックポイントの評価を行う．消費エネルギー削減効果の高いチェックポイントは，採用する DVFS 戦略によって左右される．そのため，この評価は採用する DVFS 戦略ごとに行われる．本研究では，DVFS 戦略として，最悪実行パスによる DVFS 戦略[3]，および，最頻実行パスによる DVFS 戦略[4] を CFG を使わずに適用できるように拡張したものを用いる．さらに，本研究では，このチェックポイント選定方法にグリーディ法に基づいた手法を提案する．チェックポイント候補は，DVFS に効果のある順にチェックポイントリストとしてまとめられる．これらの操作の詳細は，5 章で述べる．

プログラムには，チェックポイントリストにおける上位のチェックポイントの集合が挿入される．挿入するチェックポイントの個数を指定することで，消費エネルギー削減の有効性が高いチェックポイントの集合のみを抽出することができる．そして，採用するチェックポイントに対してチェックポイント関数が挿入される．また，チェックポイントごとに，実行トレースマイニング中の操作で得た残り実行サイクル数の見積り値を保持するテーブルを持つ．プログラムの実行がチェックポイントに到達すると，チェックポイント関数を参照する．プログラムのループ構造や関数の内部である場合は，チェックポイントの実行順を判断し，それが有効なチェックポイントの実行順であるかを判断する．この処理は，実行順を記録する変数のインクリメント，および，実行順が有効なもであるかの判断のみであり，オーバヘッドは無視できるほど小さい．また，メモリ量のオーバヘッドも，このチェックポイントの実行順を保持する変数および残り実行サイクル数の見積り値のテーブルを追加するだけである．チェックポイントの実行順が有効である場合，チェックポイント以降の残り実行

サイクル数の見積りと，デッドラインまでの時間を参照する．そして，これらの値から動作周波数を切り替えるかどうかを判断し，動作周波数を切り替える必要があれば動作周波数を切り替えて，プログラムを実行していく．

4. 実行トレースマイニング

4.1 マイニングテーブルの生成

本節では，実行トレースの取得からマイニングテーブルの作成までの過程について述べる．マイニングテーブルとは，プログラム全体の最悪実行サイクル数と，条件分岐命令における振舞いに応じた残り最悪実行サイクル数が記録されたテーブルを指す．

実行トレースは，対象となるプログラムに入力データセットを与え，サイクルレベルシミュレーションを行うことで取得する．本研究で採用する 2 種類の DVFS 戦略を効果的に行うためには，各戦略における残り実行サイクル数の見積りが大きく変動する地点を解析し，該当する地点をチェックポイントとするのが望ましい．そのため，本研究では，DVFS の対象となるプログラムで解析する情報として，実行された条件分岐命令における振舞いと，振舞いに応じた残り実行サイクル数に注目する．分岐命令に注目する理由は，条件分岐での振舞いが今後の処理内容との関連性が高く，残り実行サイクル数に影響があると考えられるためである．なお，本研究で取り扱う条件分岐命令はアセンブリレベルの命令を想定する．

実行トレースからは，**図 3** に示すように，分岐命令のアドレス，同一の条件分岐命令の実行順，振舞い，および残り実行サイクル数の一連の情報が対になって抽出される．同一の条件分岐命令の実行順とは，ループ構造や関数呼び出しなどによって，同一の分岐命令が何度も実行される場合に，それが何度目の実行かを識別するものである．振舞いとは，条件分岐命令における条件の成立，不成立を表す．本研究では，対象となる条件分岐命令のアドレスと，その直後に実行される命令のアドレスが連続している場合には，振舞いを not taken とし，連続していない場合には taken と定義する．残り実行サイクル数は，全体の実行サイクル数と注目している命令の実行サイクルの差として求まる．全体の実行サイクル数は，実行トレースの最終命令の実行サイクルから取得する．

採用する 2 種類の DVFS 戦略によって消費エネルギーを削減するためには，プログラムの残り実行サイクル数の見積りが変動する条件分岐命令のジャンプ先にチェックポイントを置くと効果的であると考えられる．なぜなら，残り実行サイクル数の見積りの変動に応じて，動作周波数を切り替えることで，デッドラインまでの残り時間を使いきれる最適な動作周波数に設定することが可能だからである．本研究では，残り実行サイクル数の見積りが変動する地点を探索するために，残り最悪実行サイクル数が減少する条件分岐命令に着目する．これは，残り最悪実行サイクル数が減少する条件分岐命令が，両 DVFS 戦略における残り実行サイクル数の見積りが変動する地点になりうると考えられるためである．そして，そのような条件分岐命令を探索するために，条件分岐命令の振舞いに応じた残り最悪実行サイクル数が記録されたマイニングテーブルを作成する．マイニングテーブルは，実行トレースから取得した条件分岐命令に関する情報をマージすることによって作成される．

マイニングテーブルの生成は，**図 2** のアルゴリズムによって行われる．アルゴリズムの処理について，図 3，**図 4** を例にして述べる．まず，図 3 の実行トレース群の中で，全体の実行サイクル数の最大値は，実行トレース 1 の 1,000 サイクルである．そのため，この 1,000 サイクルが，タスク全体の最悪実行サイクル数として，マイニングテーブルに記録される（1，2 行目）．次に，全実行トレースの中で同一の命令アドレス，実行順，および振舞いの組の中で残り実行サイクル数の最大値がマイニングテーブルに記録される．アドレスが 0x0248 で，実行順が 1 の組を例にとると，残り実行サイクル数は，実行トレース No.1 では taken で 850 サイクル，実行トレース No.2 では not taken で 400 サイクル，実行トレース No.3 では taken で 550 サイクルとなっている．そこで，最大値である実行トレース No.1 の 850 サイクルがマイニングテーブルに記録される（3，4 行目）．実行トレースから取得した条件分岐命令の情報の中には，振舞いが必ず not taken もしくは taken の片方にしかならないものが存在することがある．図 3 の実行トレース群の場合，アドレスが 0x0294 で，実行順序が 1 の組が該当する．この組は，実行トレース No.2 と No.3 に現れているが，両者とも振舞いは not taken となっている．このような振舞いが一定の条件分岐命令は，残り実行サイクル数の変動が期待できないため，図 4 のマイニングテーブルから除去される（5 行目）．以上の処理によって，図 3 の実行トレース群からは，図 4 のマイニングテーブルが生成される．

Input: 実行トレースの集合 $Traces$, **Output:** マイニングテーブル $MiningTable$
1: $Traces$ に含まれる全実行トレースの実行サイクル数を抽出
2: 1. で抽出した実行サイクル数の最大値を $MiningTable$ に記録
3: $Traces$ に含まれる全実行トレースで，アドレス，実行順，振舞いが同一の組の残り実行サイクル数を抽出
4: 3. で抽出された残り実行サイクル数の中の最大値をそれぞれ $MiningTable$ に記録
5: $MiningTable$ に記録されている組において，振舞いが not taken もしくは taken のみしかないアドレスと実行順の組を削除

図 2　マイニングテーブル生成アルゴリズム
Fig. 2　The algorithm for generating a mining table.

Execution Trace No.1		
Execution Cycles : 1000 cycles		
Address	Cond	REC
0x0248(1)	taken	850 cycles
0x0248(2)	taken	650 cycles
0x0248(3)	n_taken	450 cycles
0x026c(1)	taken	350 cycles
0x02a0(1)	n_taken	50 cycles

Execution Trace No.2		
Execution Cycles : 550 cycles		
Address	Cond	REC
0x0248(1)	n_taken	400 cycles
0x026c(1)	n_taken	300 cycles
0x0294(1)	n_taken	200 cycles
0x02a0(1)	taken	100 cycles

Execution Trace No.3		
Execution Cycles : 1000 cycles		
Address	Cond	REC
0x0248(1)	taken	550 cycles
0x0248(2)	n_taken	350 cycles
0x026c(1)	n_taken	250 cycles
0x0294(1)	n_taken	150 cycles
0x02a0(1)	n_taken	50 cycles

※REC: Remaining Execution Cycles

図 3　サイクルレベルシミュレーションで取得した実行トレース群
Fig. 3　An example of execution traces obtained via a cycle-level simulation.

Worst Case Execution Cycles : 1000cycles		
Address	RWCEC	
	n_taken	taken
0x0248(1)	400 cycles	850 cycles
0x0248(2)	350 cycles	650 cycles
0x026c(1)	300 cycles	350 cycles
0x02a0(1)	50 cycles	100 cycles

※RWCEC: Remaining Worst Case Execution Cycles

図 4　図 3 の実行トレースから作られたマイニングテーブル
Fig. 4　A mining table generated from the execution traces of Fig. 3.

実行トレースマイニングでは，生成されたマイニングテーブルと，取得した実行トレースから，チェックポイント候補が探索される．このチェックポイント候補の探索の詳細は 4.2 節で述べる．

4.2　チェックポイント候補の探索

本節では，作成したマイニングテーブルからチェックポイント候補となる地点を探索する処理について述べる．

DVFS によって消費エネルギーを削減するためには，プログラムの残り実行サイクル数の見積りが変動する条件分岐命令のジャンプ先にチェックポイントを置くと効果的であると考えられる．そこで，実行トレースマイニングでは，残り最悪実行サイクル数に着目し，この残り最悪実行サイクル数が減少する条件分岐命令を探索し，チェックポイント候補を探索する．この操作は，最初に取得した実行トレース群とマイニングテーブルから行われる．このチェックポイント候補探索アルゴリズムを**図 5** に示す．

Input: 実行トレースの集合 $Traces$，マイニングテーブル $MiningTable$
Output: 最悪実行サイクル数が減少する条件分岐命令のリスト $BranchList$

```
1:  BranchList ← φ
2:  for all trace ∈ Traces do
3:    rwcec ← MiningTable に記録されている最悪実行サイクル数
4:    trace_prev_rec ← 実行トレース trace の全実行サイクル数
5:    Num ← 実行トレース trace に含まれる条件分岐命令の個数
6:    for i = {1, 2, ..., Num} do
7:      trace_branch ← trace の i 番目にある条件分岐命令のアドレスと実行順の組
8:      if trace_branch ∈ MiningTable then
9:        trace_current_rec ← trace_branch における残り実行サイクル数
10:       executed_cycles ← trace_prev_rec − trace_current_rec
          /*trace において，前回の残り実行サイクル数 trace_prev_rec と今回の trace_branch の残り実行サイクル数 trace_current_rec から，前回から今回までの実行サイクル数 executed_cycles を算出*/
11:       trace_prev_rec ← trace_current_rec
12:       rwcec ← rwcec − executed_cycles
          /* 前回参照したマイニングテーブルによる残り最悪実行サイクル数 rwcec と，算出した executed_cycles から，trace_branch における現在の残り最悪実行サイクル数の見積りを算出 */
13:       MiningTable から，trace_branch での振舞いに該当する残り最悪実行サイクル数 table_rwcec を取得
14:     end if
15:     if table_rwcec < rwcec then
16:       rwcec ← table_rwcec
17:       if trace_branch ∉ BranchList then
18:         BranchList ← BranchList ∪ {trace_branch}
19:       end if
20:     end if
21:   end for
22: end for
```

図 5　最悪実行サイクル数が減少する条件分岐を探索するアルゴリズム
Fig. 5　The algorithm for extracting conditional branch instructions with reduced RWCEC.

図 5 のアルゴリズムについて，**図 6**，**図 7**，**図 8** を例に説明する．図 6 は本説明で対象となるプログラムの CFG であり，図 7 はプログラムにテストデータを与えて取得した実行トレースである．図 8 は図 7 から生成されるマイニングテーブルである．なお，図 6 の CFG は説明のために用いるものであり，この操作において必要なものではないことに注意されたい．

このアルゴリズムでは実行トレースマイニングで用いたすべての実行トレースに対して，実行時における残り実行サイクル数の見積りの変化を，マイニングテーブルを用いて解析

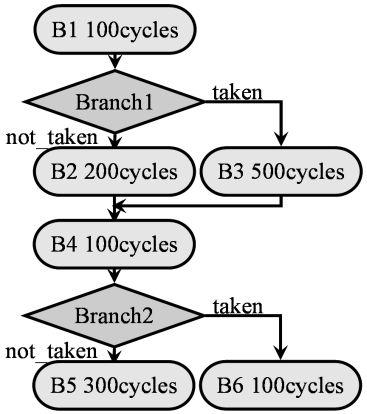

図 6　例題プログラムの CFG
Fig. 6　The CFG of an example program.

Execution Trace No.1		
Execution Cycles: 700 cycles		
Address	Condition	REC
Branch1(1)	not_taken	600 cycles
Branch2(1)	not_taken	300 cycles

Execution Trace No.2		
Execution Cycles: 800 cycles		
Address	Condition	REC
Branch1(1)	taken	700 cycles
Branch2(1)	taken	100 cycles

図 7　例題プログラムの実行トレース
Fig. 7　Execution traces of an example program.

Mining Table		
Worst Case Execution Cycles：800 cycles		
Address	RWCEC	
	not_taken	taken
Branch1(1)	600 cycles	700 cycles
Branch2(1)	300 cycles	100 cycles

図 8　図 7 の実行トレースから生成されるマイニングテーブル
Fig. 8　A mining table generated from the execution traces of Fig. 7.

する．実行トレース 1 を例にして説明する．まず，この実行トレースにおける実行開始時は，マイニングテーブルのタスク全体の最悪実行サイクル数を参照すると，残り最悪実行サイクル数は 800 サイクルであると見積もられる（3 行目）．条件分岐命令のアドレスと実行順の組は，実行トレース上の条件分岐命令の出現順に処理される（6 行目）．実行トレース 1 において，分岐 1 に到達したときを考える．このとき，実行トレース 1 の全体の実行サイクル数と，分岐 1 における残り実行サイクル数より，実行開始から 100 サイクルだけ実行されていることが分かる（10 行目）．ゆえに，この時点では残り最悪実行サイクル数は 700 サイクルと見積もられる（12 行目）．しかし，マイニングテーブルを参照すると，分岐 1 が not taken だった場合，その後の残り最悪実行サイクル数は 600 サイクルである．すなわち，分岐 1 で not taken のジャンプ先では残り最悪実行サイクル数が減少する．ゆえに，この条件分岐命令は残り最悪実行サイクル数の見積りが変動する条件分岐命令となる（15〜20 行目）．

実行トレース 1 において，分岐 2 に到達したときは，分岐 1 から 300 サイクル実行されているため，残り最悪実行サイクル数の見積りは 300 サイクルとなっている．しかし，分岐 2 で振舞いが not taken だった場合に，マイニングテーブルを参照すると，残り最悪実行サイクル数の見積りは 300 サイクルとなる．すなわち，分岐 2 で振舞いが not taken だった場合，残り最悪実行サイクル数の見積りは変動しないことが分かる．そのため，条件分岐 2 は残り最悪実行サイクル数の見積りが減少する条件分岐命令と見なさない．

チェックポイント候補の探索では，以上の操作を，入力に用いたすべての実行トレースに対して行う．そして，1 度でも残り最悪実行サイクル数が減少した条件分岐命令を *BranchList* にまとめる．チェックポイントは，採用する DVFS 戦略によって有効となるものが異なる．2 章で述べたように，最悪実行パスによる DVFS 戦略では，残り最悪実行サイクル数が減少する地点がチェックポイントに適している．そのため，本研究では，最悪実行パスによる DVFS 戦略を採用する場合，*BranchList* に含まれる条件分岐命令において，残り最悪実行サイクル数が減少する分岐先をチェックポイント候補として，5 章で使用するチェックポイント候補リスト *CandiCPList* にまとめる．一方，最頻実行パスによる DVFS 戦略を採用する場合は，*BranchList* に含まれる条件分岐命令での両方の分岐先をチェックポイント候補として，チェックポイント候補リスト *CandiCPList* にまとめる．

5. チェックポイントの選定

本章では，まず，本研究で採用する最悪実行パスを用いた DVFS 戦略と最頻実行パスを用いた DVFS 戦略におけるチェックポイントでの残り実行サイクル数の見積りについて触れる．次に，実行トレースマイニングによって探索されたチェックポイント候補から，提案する DVFS 戦略の消費エネルギー削減効果を向上させるためのチェックポイントを選定する操作について説明する．

5.1 チェックポイント間データの算出

本研究では，採用する DVFS 戦略として，最悪実行パスを用いた DVFS 戦略，および最頻実行パスを用いた DVFS 戦略の 2 種を取り上げる．チェックポイントにおける残り実行サイクル数の見積りは，採用する DVFS 戦略に依存する．そして，これらの戦略の場合の残り実行サイクル数を見積もるためには，チェックポイント間の実行サイクル数，および通

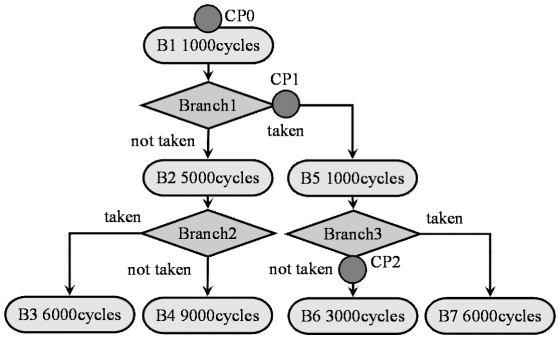

図 9　例題 CFG
Fig. 9　The CFG of an example program.

Inter-CP Data No.1	
Num = 18	
CP	REC
CP0	12000 cycles
END	0 cycles

Inter-CP Data No.2	
Num = 12	
CP	REC
CP0	15000 cycles
END	0 cycles

Inter-CP Data No.3	
Num = 14	
CP	REC
CP0	5000 cycles
CP1	4000 cycles
CP2	3000 cycles
END	0 cycles

Inter-CP Data No.4	
Num = 56	
CP	REC
CP0	8000 cycles
CP1	7000 cycles
END	0 cycles

図 10　チェックポイントの箇所のみ抽出された実行トレース群
Fig. 10　Extracted execution traces with checkpoint-related information.

過確率を求める必要がある．そこで，本節では，これらの算出方法について述べる．算出は，実行トレースマイニングで用いた実行トレースを解析することによって行われる．

実行トレースの解析によるチェックポイント間の実行サイクル数，および通過確率の算出過程を**図 9** の CFG を持つプログラム，および**図 10** の実行トレース群を用いて説明する．例として，プログラムにはチェックポイントがプログラムの実行を始める場所（CP0），分岐 1 の taken 側（CP1），および分岐 3 の not taken 側（CP2）に挿入されているものとする．図 10 の実行トレース群は，実行トレースマイニングにおいて，このプログラムに入力データを与えて得られるものから，チェックポイントが挿入されている地点の情報だけを抽出したものである．実行トレース内の Num の値は，取得できた実行トレースの数である．なお，実行トレース 1 は {B1, B2, B3}，実行トレース 2 は {B1, B2, B4}，実行トレース 3 は {B1, B5, B6}，そして実行トレース 4 は {B1, B5, B7} のノードを通過して取得できるものである．これらの実行トレースから，**図 11** のようなチェックポイントの遷移を示す有向グラフを生成する．有効グラフの生成は，ポイントの遷移の洗い出し，チェックポイント間の最悪実行サイクル数の算出，およびチェックポイント間の通過確率の算出から構成される．本説明では，プログラムの実行が終わる地点もチェックポイントと表現する．なお，これらの処理を行ううえで，プログラムの CFG を解析する必要がないことに注意されたい．すなわち，対象となるプログラムの CFG が存在しなくても，図 10 の実行トレース群を取得でき，図 11 のような有向グラフを生成することができる．

まず，ポイントの遷移の洗い出しでは，各チェックポイント間データを読み出し，連続する 2 つのポイントが注目される．これらは，前者のポイントから後者のチェックポイントへの遷移が存在することを意味する．そのため，このように連続するチェックポイントの組合せを，すべてのチェックポイント間データから抽出し，図 11 のようにまとめる．

次に，チェックポイント間の最悪実行サイクル数を算出する．各実行トレースにおいて，チェックポイント間の実行サイクル数は，連続するポイントの残り実行サイクル数の差で表現される．そこで，すべてのチェックポイント間データに含まれる，2 つのチェックポイント間の実行サイクル数を求め，その最大値を図 11 のように記録する．たとえば，チェックポイント 0 とプログラムの終端ポイントまでの遷移の場合，実行トレース 1 では 12,000 サイクル，実行トレース 2 では 15,000 サイクル存在する．この場合，後者の 15,000 サイクルが記録される．

最後に，チェックポイント間の遷移確率を算出する．チェックポイント X とチェックポイント Y 間の通過確率は，チェックポイント X に到達する実行トレースの総数に対する，チェックポイント X からチェックポイント Y に遷移する実行トレースの総数の比で表される．たとえば，チェックポイント 1 から終端ポイントへの遷移確率は，チェックポイント 1 に到達する実行トレースは 70 個存在し，そこから終端ポイントへ到達する実行トレースが 56 個存在するため，0.8 と表現される．この操作をすべてのチェックポイント間の遷移に対して行い，すべての遷移確率が求められる．

チェックポイントでは，次の動作周波数を計算し，動作周波数の切替えを判断するために実行サイクル数のオーバヘッドがかかる．さらに，動作周波数を切り替えた場合に，さらなる時間的な遅延も発生する．ゆえに，各 DVFS 戦略に対して，チェックポイントにおける残り実行サイクル数を求めるためには，これらのオーバヘッドを考慮しなければならない．そこで，有向グラフにおけるチェックポイント間の最悪実行サイクル数に対して，チェックポイントの実行サイクル数を付加することで，オーバヘッドを考慮する．

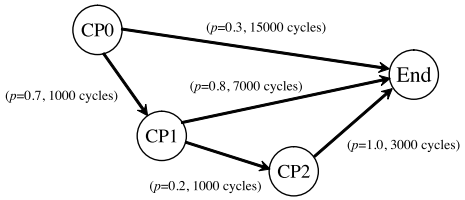

図 11　チェックポイントのオーバヘッドを考慮しない有向グラフ
Fig. 11　A CP directed graph without considering overheads of checkpoints.

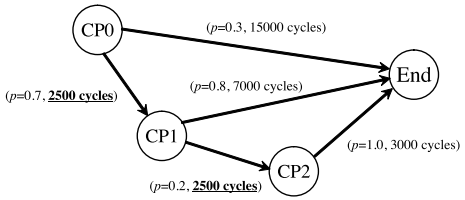

図 12　チェックポイントのオーバヘッドを考慮した有向グラフ
Fig. 12　A CP directed graph with considering overheads of checkpoints.

チェックポイントの実行サイクル数は，チェックポイント内で動作周波数の計算にかかる実行サイクル数と，動作周波数切替えにともなう遅延時間を実行サイクル数に換算されたものの総和である．後者の換算は，動作周波数切替えにともなう遅延時間と，設定できる動作周波数の最大値の積を求めることで行われる．たとえば，チェックポイント内で動作周波数の計算にかかる実行サイクル数が 500 サイクル，遅延時間が 10 μs，プロセッサの設定できる動作周波数の最大値が 100 MHz であるとする．このとき，図 11 におけるチェックポイント 1 とチェックポイント 2 の間の最悪実行サイクル数は，$1{,}000 + 500 + 100\,\text{MHz} \times 10\,\mu\text{s} = 2{,}500$ サイクルと計算される．なお，終端ポイントでは動作周波数を切り替えることなく，プログラムを終えるため，チェックポイントと終端ポイント間の最悪実行サイクル数については，この補正は行われない．ゆえに，図 11 におけるチェックポイント 2 と終端ポイントの間では，最悪実行サイクル数は 3,000 サイクルのままである．図 11 のチェックポイントの遷移において，実行サイクル数に修正を加えたものを**図 12** に示す．

複雑な構造を持つプログラムの場合，通過するチェックポイントに順序関係が成り立たない場合がある．すなわち，ある入力データではチェックポイント 1 の後にチェックポイント 2 を通過しても，別の入力データではチェックポイント 2 を通過した後にチェックポイント 1 を通過することがある．本研究では，そのような状況を回避するために，まず，残り実行サイクル数の見積り値が大きいチェックポイントほど，チェックポイントの番号を小さくする．次に，DVFS 実行時において，チェックポイントを通過したときに，通過したチェックポイントの番号を記憶する．そして，もし新たに通過したチェックポイントの番号が，前回通過したチェックポイントの番号より小さい場合は，そのチェックポイントを無視する．この規則によって，5.2 節で述べる残り実行サイクル数の見積りの計算が可能となる．

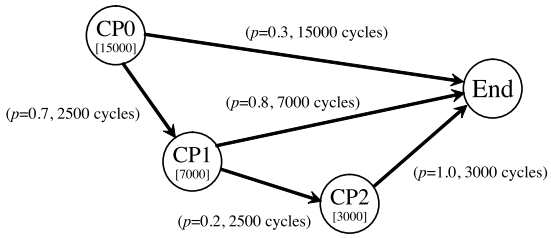

図 13　最悪実行パスを用いた DVFS 戦略での残り実行サイクル数の見積り
Fig. 13　The estimation of remaining execution cycles in the worst case execution path DVFS.

5.2　残り実行サイクル数の見積りと動作周波数の切替え

5.2.1　最悪実行パスを用いた DVFS 戦略の場合

最悪実行パスを用いた DVFS 戦略とは，プログラムの実行が終わるまでの最悪実行サイクル数を用いた手法である．この戦略は，プログラム終了までの実行サイクル数の最大値を用いるため，デッドライン制約を保証しやすいという利点が存在する．しかし，残り実行サイクル数を多く見積もり，高めの動作周波数で実行する可能性が高い．そのため，消費エネルギー削減効果が小さいという欠点が存在する．

最悪実行パスを用いた DVFS 戦略では，チェックポイントにおける残り実行サイクル数の見積りとして，終端ポイントまで至るすべての経路の実行サイクル数の最大値が選択される．**図 13** のチェックポイント 0 では，終端ポイントまでの経路として，{CP0, End}, {CP0, CP1, End}，および {CP0, CP1, CP2, End} が考えられる．それぞれの最悪実行サイクル数は，15,000 サイクル，9,500 サイクル，そして 8,000 サイクルである．よって，チェックポイント 0 では，残り最悪実行サイクル数として 15,000 サイクルが選ばれる．チェックポイントのノードにある括弧内の数値は，そのチェックポイントにおける残り最悪実行サイクル数である．

実行時における，チェックポイントでの動作周波数切替えアルゴリズムについて述べる．動作周波数切替えアルゴリズムを**図 14** に示す．プログラムの実行がチェックポイントに到達すると，まず，現在の時刻 t が参照される（1 行目）．次に，デッドライン D までの時間とチェックポイント CP_i における残り最悪実行サイクル数 $N_{worst}(CP_i)$ から最適な動作周波数が計算される（2 行目）．このとき，動作周波数を切り替える際の遅延時間 Δt も考慮する．続いて，算出された動作周波数を設定可能な動作周波数に修正する．これは，商用のプロセッサでは，設定できる動作周波数は離散的で限られているため，算出された動作周波

Input: 現在のチェックポイント CP_i，**Output:** なし
1: 現在時刻 t を参照
2: $f_{tmp} \leftarrow \dfrac{N_{worst}(CP_i)}{D - t - \Delta t}$
3: $f_{next} \leftarrow \begin{cases} f_0 & (f_{tmp} \leq f_0) \\ f_i & (f_{i-1} < f_{tmp} \leq f_i, i \in \{1, 2, \cdots, n\}) \\ f_n & (\text{otherwise}) \end{cases}$
4: 現在の動作周波数が f_{next} でなければ，動作周波数を f_{next} に切替え

図 14　最悪実行パスを用いた DVFS 戦略での動作周波数切替えアルゴリズム
Fig. 14　The algorithm for switching frequency in the worst case execution path DVFS.

数 f_{tmp} に切り替えられるとは限らないためである．そこで，設定できる動作周波数を値が小さいものから $\{f_0, f_1, \cdots, f_n\}$ とすると，動作周波数は f_{tmp} 以上で，かつ，設定可能な動作周波数で最小のものが選ばれる．f_{tmp} 以上のものがない場合は，設定可能な最大の動作周波数 f_n が選ばれる（3 行目）．

5.2.2　最頻実行パスを用いた DVFS 戦略の場合

最悪実行パスを用いた DVFS 戦略では，残り実行サイクル数が悲観的な値となることから，動作周波数は高く設定されることが多く，消費エネルギーの削減効果は小さいと考えられる．そこで，この悲観さを緩和した方法として，最頻実行パスを用いた DVFS 戦略を取り上げる．

最頻実行パスを用いた DVFS 戦略とは，プログラム終了に至るまでのチェックポイントの遷移の中で，最も確率の高いチェックポイントの遷移の実行サイクル数を用いた手法である．この戦略で用いられる実行サイクル数を最頻実行サイクル数と定義する．一般的なプログラムでは，最悪実行パスを実行する確率は低いため，通過確率の高い遷移の実行サイクル数を見積もることで悲観さを緩和できることがこの手法の利点といえる．ゆえに，最悪実行パスの DVFS と比較して，動作周波数を低めに設定できることが期待でき，平均消費エネルギーをより削減できる可能性が高いと考えられる．**図 15** のチェックポイント 0 では，終端ポイントまでの経路の中で，{CP0, CP1, End} が最も通過する確率が高い．よって，チェックポイント 0 では，残り最頻実行サイクル数として，{CP0, CP1, End} の経路の 9,500 サイクルが選ばれる．チェックポイントのノードにある括弧内の数値は，そのチェックポイントにおける残り最頻実行サイクル数である．

この戦略の欠点として，最頻実行パスと比較して多くの実行サイクル数を持つ実行パスを通過した場合に，デッドラインミスが発生する可能性があることがあげられる．図 15 のようなチェックポイントの遷移を持つプログラムを例として説明する．説明のために，プログラムのデッドラインまでの時間が 190 μs と仮定する．チェックポイント 0 では，残り実行サイクル数として 9,500 サイクルが見積もられる．そのため，プログラム実行の際に，チェックポイント 0 で動作周波数は 50 MHz に設定される．しかし，チェックポイント 0 からチェックポイント 1 を通過せずに End に到達してしまう入力データが与えられた場合，15,000 サイクルを 50 MHz で実行するため，300 μs の時間がかかる．ゆえに，デッドラインの 190 μs を超えてしまうために，デッドラインミスが発生する．

本研究では，以上のデッドラインミスの対策をするために，各チェックポイントに中間デッドラインを設け，それを用いた動作周波数の切替えアルゴリズムを採用する．このアルゴリズムでは，最頻実行サイクル数を用いて動作周波数を算出した後に，その動作周波数が後続のチェックポイントの中間デッドラインを守られるかどうかを判断する．そして，守られない場合には，該当するチェックポイントの中間デッドラインを用いて動作周波数の再計算が行われる．

このアルゴリズムを実現するために，静的に行われる処理について述べる．まず，各チェックポイントにおける中間デッドラインを算出する．中間デッドラインとは，チェックポイント以降，設定可能な動作周波数の最大値で実行を続ければデッドラインミスを起こさない時刻であり，式 (1) で与えられる．プログラムのデッドラインの値 d_{CP_i} はチェックポイント CP_i における中間デッドラインであり，$N_{worst}(CP_i)$ はチェックポイント CP_i 以降の残り最悪実行サイクル数，D はデッドラインの時刻，そして f_{max} は設定可能な動作周波数の最大値である．なお，終端ポイントの中間デッドラインは，そのプログラムのデッドラインの値そのものが割り当てられる．図 15 の有向グラフを持つプログラムに対して，設定可能な動作周波数の最大値が 100 MHz であれば，各チェックポイントの中間デッドラインは，**図 16** のように与えられる．さらに，動作周波数の再計算の際に必要な情報として，各チェックポイントに対して，次に到達するチェックポイントと，それらまでの最悪実行サイクル数の情報を付与する．

$$d_{CP_i} = D - \frac{N_{worst}(CP_i)}{f_{max}} \tag{1}$$

実行時の振舞いについて述べる．実行時における，動作周波数切替えアルゴリズムを**図 17** に示す．まず，プログラムの実行がチェックポイント CP_i に到達すると，現在の時刻 t が参照される（1 行目）．次に，デッドライン D までの時間とチェックポイント CP_i における残り最頻実行サイクル数 $N_{average}(CP_i)$ から最適な動作周波数が計算される（2，3 行目）．

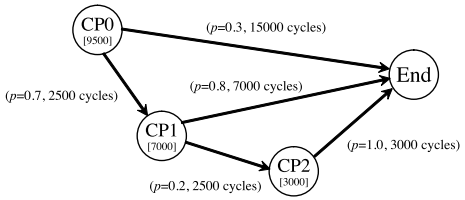

図 15　最頻実行パスを用いた DVFS 戦略での残り実行サイクル数の見積り
Fig. 15　The estimation of remaining execution cycles in the average case execution path DVFS.

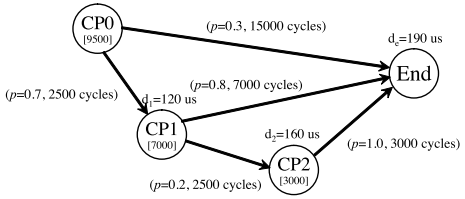

図 16　中間デッドラインの算出
Fig. 16　Calculation for middle deadline.

Input: 現在のチェックポイント CP_i，**Output:** なし

1: 現在時刻 t を参照

2: $f_{tmp} \leftarrow \dfrac{N_{average}(CP_i)}{D - t - \Delta t}$

3: $f_{next} \leftarrow \begin{cases} f_0 & (f_{tmp} \le f_0) \\ f_i & (f_{i-1} < f_{tmp} \le f_i, i \in \{1, 2, \cdots, n\}) \\ f_n & (\text{otherwise}) \end{cases}$

4: **for all** $CP_{i'} \in CPNext(CP_i)$ **do**

5:　**if** $d_{CP_{i'}} - t < \dfrac{N'_{worst}(CP_i, CP_{i'})}{f_{next}}$ **then**

6:　　$f_{tmp} \leftarrow \dfrac{N'_{worst}(CP_i, CP_{i'})}{d_{CP_{i'}} - t - \Delta t}$

7:　　$f_{next\_tmp} \leftarrow \begin{cases} f_0 & (f_{tmp} \le f_0) \\ f_i & (f_{i-1} < f_{tmp} \le f_i, i \in \{1, 2, \cdots, n\}) \\ f_n & (\text{otherwise}) \end{cases}$

8:　　**if** $f_{next} < f_{next\_tmp}$ **then**

9:　　$f_{next} \leftarrow f_{next\_tmp}$

10:　　**end if**

11:　**end if**

12: **end for**

13: 現在の動作周波数が f_{next} でなければ，動作周波数を f_{next} に切替え

図 17　最頻実行パスを用いた DVFS 戦略の動作周波数切替えアルゴリズム
Fig. 17　The algorithm for switching frequency in the average case execution path DVFS.

そして，この動作周波数で実行した場合に，後続のチェックポイントの中間デッドラインが満たされるかどうかを確認する（4 行目〜9 行目）．ここで，$CPNext(CP_i)$ はチェックポイント CP_i の次に到達しうるチェックポイントの集合であり，$N'_{worst}(CP_i, CP_{i'})$ はチェックポイント CP_i と $CP_{i'}$ の間の最悪実行サイクル数である．すなわち，後続のチェックポイントまでの残り時間内に，そのチェックポイントに到達できるかどうかを判断する（5 行目）．もし，到達できない場合には，そのチェックポイントの中間デッドラインに間に合うような動作周波数に修正する（7，8 行目）．以上の処理を，$CPNext(CP_i)$ に含まれるすべてのチェックポイントに対して行う．

実行時の振舞いを，図 16 の有向グラフを持つプログラムの場合について述べる．仮定として，デッドラインまでの時間が 190 μs，設定できる周波数が {10, 20, ..., 100} MHz，そして簡単のために周波数切替えにともなう遅延時間 Δt が無視できる場合を考える．実行時には，チェックポイント 0 において，残り最頻実行サイクル数の 9,500 サイクルと，190 μs から f_{next} として 50 MHz が計算される（1〜3 行目）．次に，チェックポイント 0 から遷移する可能性のあるチェックポイントの中間デッドラインと，そこまでの最悪実行サイクル数を参照し，算出された f_{next} で中間デッドラインを満たすかどうかを確認する（5 行目）．本例の場合，チェックポイント 0 から終端ポイントへの遷移すると，中間デッドラインを守ることができない．その場合，終端ポイントの中間デッドラインを守れるように，周波数の修正を行う（6〜9 行目）．ゆえに，チェックポイント 0 では，チェックポイント 0 から終端ポイントまでの 15,000 サイクルと，中間デッドラインまでの 190 μs の時間から 80 MHz に修正される．

本例では，デッドラインミス対策によって，最終的に設定される動作周波数は最悪実行パスを用いた DVFS 戦略と同等となる．しかし，チェックポイントを挿入する地点を変えることで，最悪実行パスを用いた DVFS 戦略より消費エネルギーを削減できる可能性がある．

5.3　グリーディ法によるチェックポイントの順位付け

本節では，実行トレースマイニングによって抽出したチェックポイント候補から，DVFS の効果を向上させるチェックポイントを選定する過程について説明する．

チェックポイントでは，動作周波数の切替えにともなう遅延時間および消費エネルギーのオーバヘッドが存在する．そのため，プログラム中にチェックポイントが過剰に挿入されると，消費エネルギーの削減効果が減少する可能性がある．ゆえに，チェックポイント候補から消費エネルギー削減効果の高いチェックポイントの組合せが選定されることが望ましい．しかし，全数探索によって最適な組合せを探索することは計算複雑さの観点から現実的ではない．全数探索の場合，チェックポイント候補の数を N 個とすると，最適なチェックポイントの組合せを探索するには $O(2^N)$ の計算量が必要になる．そのため，本研究では，この計算量を緩和するために，グリーディ法に基づいた選定手法を採用する．

```
Input: チェックポイント候補リスト CandiCPList
Output: 順位付きチェックポイント RankedCP_1, RankedCP_2, ..., RankedCP_n
 1: RankedCPList ← φ
 2: InsertedCPList ← φ
 3: fixed_num ← 0
 4: candi_num ← CandiCPList に含まれるチェックポイント候補の数
 5: while fixed_num < candi_num do
 6:    energy_min ← 0
 7:    for all CP_tmp ∈ CandiCPList do
 8:       InsertedCPList ← RankedCPList ∪ {CP_tmp}
 9:       energy ← Evaluate_Energy(InsertedCPList)
          /* InsertedCPList が挿入されたプログラムに対して，入力データを与えて DVFS を試行して消費エ
          ネルギーを算出 */
10:       if energy_min > energy or energy_min = 0 then
11:          energy_min ← energy
12:          CP_best ← CP_tmp
13:       end if
14:    end for
15:    if energy_min = 0 then
16:       break
17:    else
18:       RankedCPList ← RankedCPList ∪ {CP_best}
19:       CandiCPList ← CandiCPList \ {CP_best}
20:       fixed_num ← fixed_num + 1
21:       RankedCP_fixed_num ← CP_best
22:    end if
23: end while
```

図 18　チェックポイント順位付けアルゴリズム
Fig. 18　The algorithm for ranking checkpoints.

グリーディ法によるチェックポイントの順位付けアルゴリズムを**図 18** に示す．このアルゴリズムでは，実行トレースマイニングで求められたチェックポイント候補リスト $CandiCPList$ が入力として与えられ，順位が付けられたチェックポイントのリストを意味する変数の集合 $RankedCP_1, RankedCP_2, \ldots, RankedCP_n$ が出力される．アルゴリズムの処理として，まず，すでに順位付けられたチェックポイント $RankedCPList$ と，$CandiCPList$ に含まれているものの中の 1 つを有効にする（8 行目）．次に，DVFS を試行し，消費エネルギーを算出する（9 行目）．$Evaluate\_Energy(CPList)$ は，引数として与えられるリスト $CPList$ に含まれるチェックポイントが挿入されたプログラムに対して，入力データを与えて DVFS を試行したときの消費エネルギーを返す関数である．消費エネルギーの算出は，与えられるすべての入力データに対して行われ，$Evaluate\_Energy(CPList)$ はそれらの合計値を出力する．ただし，この算出においてデッドラインミスが起きる入力データが存在した場合には，消費エネルギーとして 0 が出力されるものとする．最後に，新たに有効にしたチェックポイントの中で，最も消費エネルギーが少ないものを $RankedCP_{fixed\_num}$ に記録する（21 行目）．また，そのチェックポイントは，次のチェックポイント選定のために，$RankedCPList$ へ追加し $CandiCPList$ から除外する（18，19 行目）．この処理を，すべてのチェックポイント候補に順位が付けられる，あるいは，新たにチェックポイントを有効にすると必ずデッドラインミスが発生してしまうまで行う．このとき，チェックポイントの順位付け操作の計算量は $O(N^2)$ と評価できる．

6. 評価実験

6.1 実験環境

評価実験では，提案手法を用いて DVFS を行った場合の消費エネルギー量で評価した．実験対象のプログラムとして，入力データによって処理時間が変動する MediaBench [14] の jpeg エンコード（cjpeg）およびデコード（djpeg）のプログラムを採用した．本実験では，両プログラムに対して 100 種類の同じサイズの画像を用意し，入力データセットとして用いた．入力データのサイズは，縦横の大きさを 100×50 pixel とした．サイクルレベルシミュレータとして SimpleScalar/ARM [15] を用いた．SimpleScalar/ARM とは，組込みシステムのプロセッサとして広く使われている ARM7TDMI [16] の命令セットシミュレータである．

実験の仮定について述べる．まず，設定できる動作周波数に関する仮定について説明する．プロセッサが設定できる動作周波数は，10 MHz 刻みで $\{10, 20, \cdots, 100\}$ MHz とした．また，チェックポイントのオーバヘッドを 1,000 サイクル，動作周波数切替えにかかる遅延時間を 300 μs と仮定した．なお，チェックポイントの実行順を判断するためのオーバヘッドは無視できるほど小さいため，考慮していない．

次に，デッドラインまでの時間について述べる．デッドラインまでの時間は式 (2) で与えられるものとした．

$$Deadline = \frac{WCET}{1-\alpha} \tag{2}$$

$WCET$ とは，設定可能な動作周波数の最大値である 100 MHz で実行した際の最悪実行時間である．本実験における各ベンチマークの $WCET$ は，cjpeg では 45,594 μs，djpeg

では 12,747 μs であった．ここで，α はスラック係数と呼ばれる実数値である．本論文におけるスラック係数とは，タスクのデッドラインまでの実行時間の余裕度を表す．つまり，スラック係数が大きければ大きいほど，デッドラインまでの余裕時間が増えることを意味する．本実験では，デッドラインまでの時間として，式 (2) での α に対して，0，0.1，0.2，0.3 を代入した値を用いた．消費エネルギーは，リーク電流による消費エネルギーは無視できるものとし，動作周波数 f_i における実行サイクル数 N_i の重み付き和 $\sum_i f_i^2 \times N_i$ として算出した[1)]．

実験では，最悪実行パスによる Intra-task DVFS 戦略，および最頻実行パスによる Intra-task DVFS 戦略による消費エネルギーの削減量を比較した．5 章で述べたとおり，チェックポイントにおける処理には実行時間と消費エネルギーのオーバヘッドがかかる．そのため，実験では，提案した 2 つの手法の比較の際，チェックポイントの個数をパラメータとし，個数を変化させて消費エネルギーの削減量を評価した．これにより，チェックポイントの個数による消費エネルギーのオーバヘッドへの影響，および，リアルタイム性（デッドラインミス）への影響を確認する．さらに，提案した 2 つの手法に加えて，Highest Speed と Static DVFS 戦略での消費エネルギーの削減量も取り上げた．Highest Speed とはプログラムの実行開始から実行終了まで最大動作周波数である 100 MHz で実行し続けるものである．Static DVFS とは，実行開始時に固定の動作周波数を与えて最後まで実行し続ける手法である．Static DVFS で与えられる動作周波数は，設定可能な動作周波数で，かつデッドラインを保証することのできる最小の動作周波数である．Static DVFS は，実用システムでもよく適用されている現実的な手法である．この両者は，プログラムの途中で動作周波数を切り替えることはなく，つねに同じ動作周波数で実行が続けられる．なお，両者の値は，式 (2) における α が 0 のときには等しくなる．なお，消費エネルギーは，プログラムに対して実行トレースマイニングで用いた入力データと同等のものを与えて算出した．

6.2 実験結果

提案手法による cjpeg および djpeg に対して DVFS を適用した場合の消費エネルギーをグラフ化したものを，**図 19** と**図 20** に示す．チェックポイントは，挿入しない場合，チェックポイントリストに記録されているものの中から上位の 20 個，50 個，100 個，150 個を挿入した場合，そして，デッドラインミスが発生しない限界の個数を挿入した場合の計 7 種とした．チェックポイントを挿入すると，チェックポイントのオーバヘッドによって，実行サイクル数が増加する．そのため，チェックポイントを過剰に挿入すると，デッドラインミスの対策を施していたとしても，デッドラインミスが起きてしまう可能性がある．そこで，

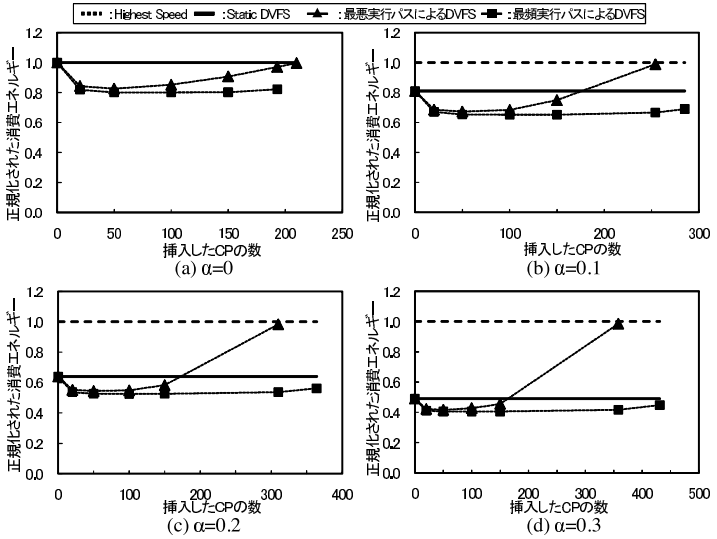

図 19　cjpeg における正規化された消費エネルギー
Fig. 19　Normalized energy results of cjpeg benchmark.

チェックポイントの順位付けにおいて DVFS を試行する際に，新しくチェックポイントを挿入すると必ずデッドラインミスが発生してしまうチェックポイントの個数を，デッドラインミスが発生しない限界の個数として取り上げた．なお，実行トレースマイニングによって得られたチェックポイント候補の総数は，最悪実行パスを用いた DVFS の場合，cjpeg では 389 個，djpeg では 173 個であった．また，最頻実行パスを用いた DVFS の場合，cjpeg では 778 個，djpeg では 346 個であった．

実験結果の詳細について述べる．まず，すべての状況において Highest Speed と提案手法による DVFS を比較すると，最悪実行パスによる DVFS および，最頻実行パスによる DVFS の方が，ともに消費エネルギーを多く削減できていることが分かる．さらに，デッドラインまでの時間が長いほどこれらの手法の消費エネルギーが多く削減できていることが確認できる．次に，Static DVFS との比較について述べる．提案した 2 つの DVFS 手法による消費エネルギーと，Static DVFS を行った場合の消費エネルギーと比較すると，提案手法による DVFS の方が，消費エネルギーが多く削減されていることが確認できる．特に，djpeg で $\alpha = 0$ の場合に着目すると，チェックポイントの個数が 50 の場合が最大であり，

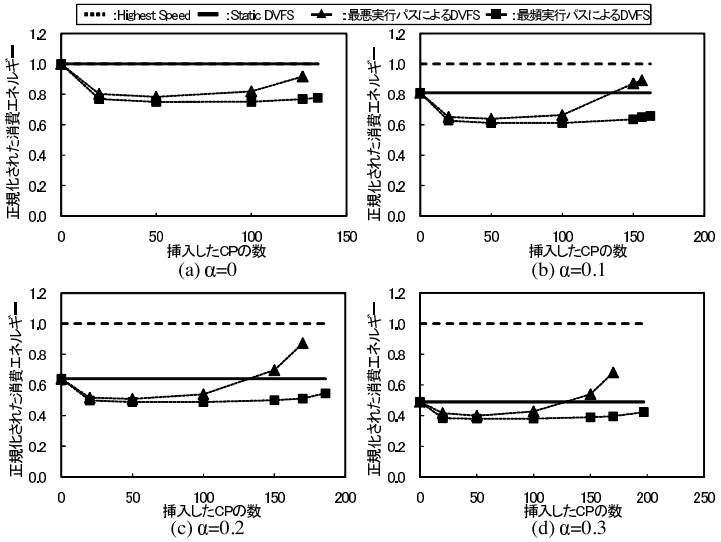

図 20　djpeg における正規化された消費エネルギー
Fig. 20　Normalized energy results of djpeg benchmark.

最悪実行パスによる DVFS 戦略では 21.6%，最頻実行パスによる DVFS 戦略では 25.0%の消費エネルギーが削減できている．続いて，最悪実行パスによる DVFS 戦略と，最頻実行パスによる DVFS 戦略による消費エネルギー量を比較する．両者を比較すると，ほぼすべての場合において，最頻実行パスによる DVFS 戦略の方が多くの消費エネルギーを削減できていることが確認できる．例として，チェックポイントの個数が 50 個の場合を比較すると，cjpeg では 2.6%から 3.3%，djpeg では 4.0%から 4.7%，最頻実行パスによる DVFS 戦略の方が消費エネルギーを多く削減できていることが分かった．最後に，チェックポイントの個数に応じた消費エネルギーの削減量について触れる．チェックポイントはある一定の個数までならば，個数を増やすと同時に消費エネルギーも削減できることが確認できた．逆に，過剰にチェックポイントを挿入すると，個数が増えると同時に消費エネルギーが増加することも分かった．なお，すべての戦略において，チェックポイントの個数を変化させても，デッドラインミスは確認されなかった．

6.3 考　　察

本節では，実験結果に対する考察を述べる．Highest Speed と比較して，提案手法による DVFS は消費エネルギーを多く削減できている．また，デッドラインまでの時間が長いほど，消費エネルギーが多く削減できている．この傾向は，デッドラインまでの時間と余裕時間も長くなり，動作周波数を大きく下げられるようになったためと考えられる．

次に，Static DVFS との比較について述べる．提案した 2 つの DVFS 手法による消費エネルギーと，Static DVFS を行った場合の消費エネルギーと比較すると，提案手法による DVFS の方が，消費エネルギーが多く削減されていることが確認できる．これは，プログラムの途中に挿入されたチェックポイントによって，細かい粒度で動作周波数の切替えを行うことができたためであると考えられる．しかし，最悪実行パスによる DVFS 戦略では，チェックポイントを過剰に挿入した場合に，Static DVFS と比較して，消費エネルギーが多くなることが確認できる．これは，多くのチェックポイントを挿入することによって，チェックポイントにおける残り最悪実行サイクル数が増大し，設定される動作周波数が大きくなったことが原因と考えられる．

続いて，チェックポイントの個数による消費エネルギーの削減量について説明する．例として，cjpeg における $\alpha = 0$ の場合に着目する．提案した 2 つの DVFS 戦略の場合，チェックポイントの個数が 50 まで増加すると同時に，消費エネルギーの削減量も増加していることが分かる．この理由は，チェックポイントを多く挿入することで，より細かい間隔で低い動作周波数に切り替えられるためである．しかし，チェックポイントがさらに多く挿入される場合の消費エネルギーは，チェックポイントの数が 50 個の場合と比べて増加していることが分かる．これは，チェックポイントのオーバヘッドが大きくなり，実行サイクル数が増加して動作周波数に影響を与えたためと考えられる．つまり，挿入するチェックポイントの個数を増やすことが必ずしも消費エネルギー削減につながらず，消費エネルギーを最適化できるチェックポイントの個数を変数として決定できる可能性を示している．

最後に，採用する DVFS 戦略による消費エネルギーの削減効果を考察する．最悪実行パスによる DVFS 戦略と，最頻実行パスによる DVFS 戦略による消費エネルギー量を比較すると，後者の方が消費エネルギーを多く削減できている．これは，通過確率が相対的に高くない最悪実行サイクル数を持つ実行パスより，最も通過する確率の高い実行パスに着目して動作周波数を設定した方が消費エネルギーを多く削減できるためと考えられる．しかし，プログラム内のパスの実行サイクル数や通過確率によって，最頻実行パスによる DVFS 戦略より，最悪実行パスによる DVFS 戦略の方が多くの消費エネルギーを削減できる例も報告されている[6]．そして，どのようなプログラムに適用しても，消費エネルギーが最適になるような DVFS 戦略について，今後考えていく必要がある．

7. おわりに

本研究では，シングルタスクで構成されるプログラムに対して，タスク内 DVFS によって，デッドラインの制約を保証しつつ，プロセッサの消費エネルギーの削減を目指した．そのために，チェックポイント候補を求めるための実行トレースマイニングと，求められた複数のチェックポイント候補から，特に効果の高いものを絞るチェックポイント選定法を提案した．

実行トレースマイニングでは，実行トレースを活用することで，実行時のタスクの振舞いをより正確に解析することが可能となる．同時に，設計時に容易に取得することができる実行トレースを活用することで，プログラムの CFG の解析が不要となる．チェックポイント選定法では，チェックポイントを挿入することによって発生するオーバヘッドを考慮したうえで，消費エネルギーの削減に寄与するチェックポイントを選定する．評価実験では，設定できる動作周波数が離散的である想定，かつ，動作周波数切替えにともなうオーバヘッドが存在するという前提の下で，最悪実行パスによる DVFS 戦略と最頻実行パスによる DVFS 戦略に提案手法を適用した．そして，チェックポイントの個数を適切な数にすることで，Static DVFS と比較すると，前者では最大 21.7%，後者では最大 25.0%の消費エネルギーを下げられることが確認できた．

今後の課題としては，最適なチェックポイントの個数を探索できるアルゴリズムの開発，および，より消費エネルギーを削減できるような DVFS 戦略の提案と適用があげられる．

謝辞　本研究において，多くのご指導をいただきました菊地武彦氏，横山哲郎博士に深く感謝いたします．本研究の一部は，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) Shin, D. and Kim, J.: Optimizing Intratask Voltage Scheduling Using Profile and Data-Flow Information, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, Vol.26, No.2, pp.369–385 (online), DOI:10.1109/TCAD.2006.883928 (2007).
2) Azevedo, A., Issenin, I., Cornea, R., Gupta, R., Dutt, N., Veidenbaum, A. and Nicolau, A.: Profile-Based Dynamic Voltage Scheduling Using Program Checkpoints, *Design, Automation and Test in Europe*, pp.168–175, IEEE Computer Society (2002).
3) Shin, D., Kim, J. and Lee, S.: Intra-Task Voltage Scheduling for Low-Energy, Hard Real-Time Applications, *IEEE Design & Test of Computers*, Vol.18, No.2, pp.20–30 (2001).
4) Shin, D. and Kim, J.: A Profile-Based Energy-Efficient Intra-Task Voltage Scheduling Algorithm For Real-Time Applications, *International Symposium on Low Power Electronics and Design*, pp.271–274, ACM (online), DOI:http://doi.acm.org/10.1145/383082.383162 (2001).
5) Kumar, G. and Manimaran, G.: An Intra-task DVS Algorithm Exploiting Program Path Locality for Real-Time Embedded Systems, *High Performance Computing HiPC 2005*, Bader, D., Parashar, M., Sridhar, V. and Prasanna, V. (Eds.), Lecture Notes in Computer Science, Vol.3769, pp.225–234, Springer Berlin/Heidelberg (2005).
6) Seo, J., Kim, T. and Chung, K.-S.: Profile-Based Optimal Intra-Task Voltage Scheduling for Hard Real-Time Applications, *Annual Conference on Design Automation*, pp.87–92, ACM (online), DOI:http://doi.acm.org/10.1145/996566.996597 (2004).
7) Henkel, J. and Parameswaran, S. (Eds.): *Designing Embedded Processors: A Low Power Perspective*, Springer (2007).
8) Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J. and Stenström, P.: The worst-case execution-time problem – overview of methods and survey of tools, *ACM Trans. Embededd Computing Systems* (*TECS*), Vol.7, pp.36:1–36:53 (online), DOI:http://doi.acm.org/10.1145/1347375.1347389 (2008).
9) ガイオ・テクノロジー カバレッジマスター winAMS，入手先⟨http://www.gaio.co.jp/product/dev_tools/pdt07_winams.html⟩.
10) Ball, T. and Larus, J.R.: Efficient Path Profiling, *Proc. 29th Annual International Symposium on Microarchitecture*, pp.46–57 (1996).
11) Larus, J.R.: Whole Program Paths, *Proc. SIGPLAN '99 Conference on Programming Languages Design and Implementation*, Atlanta, GA (1999).
12) Xu, R., Xi, C., Melhem, R. and Moss, D.: Practical PACE for Embedded Systems, *International Conference on Embedded Software*, pp.54–63, ACM (online), DOI:http://doi.acm.org/10.1145/1017753.1017767 (2004).
13) Xian, C. and Lu, Y.-H.: Dynamic voltage scaling for multitasking real-time systems with uncertain execution time, *Proc. 16th ACM Great Lakes symposium on VLSI*, *GLSVLSI '06*, pp.392–397, ACM, New York, NY, USA (online), DOI:http://doi.acm.org/10.1145/1127908.1127998 (2006).

14) Lee, C., Potkonjak, M. and Mangione-Smith, W.H.: MediaBench: A tool for evaluating and synthesizing multimedia and communicatons systems, *Proc. 30th Annual ACM/IEEE International Symposium on Microarchitecture*, MICRO 30, Washington, DC, USA, IEEE Computer Society, pp.330–335 (online) (1997), available from ⟨http://portal.acm.org/citation.cfm?id=266800.266832⟩.
15) SimpleScalar LLC, available from ⟨http://www.simplescalar.com/⟩.
16) ARM7TDMI ARM Processor, available from ⟨http://www.arm.com/products/CPUs/ARM920T.html⟩.

(平成 23 年 3 月 15 日受付)
(平成 23 年 9 月 12 日採録)

立松　知紘

2009 年 3 月名古屋大学工学部卒業．2011 年 3 月名古屋大学大学院情報科学研究科博士前期課程修了．修士（情報科学）．同年 4 月より西日本旅客鉄道株式会社に勤務．

高瀬　英希（学生会員）

2007 年 3 月名古屋大学工学部卒業．2009 年 3 月名古屋大学大学院情報科学研究科博士前期課程修了．2009 年 4 月より名古屋大学大学院情報科学研究科博士後期課程，同じく，日本学術振興会特別研究員 DC．現在に至る．情報処理学会 2007 年度コンピュータサイエンス領域奨励賞，平成 21 年度 IPSJ 論文船井若手奨励賞受賞．コンパイラ技術，組込みシステムの消費エネルギー最適化等の研究に従事．修士（情報科学）．

曾　　剛（正会員）

2006 年千葉大学大学院自然科学研究科博士課程修了．同年名古屋大学大学院情報科学研究科附属組込みシステム研究センター研究員，2008 年同特任助教を経て，2010 年 4 月より同大学院工学研究科講師．組込みシステム設計，エネルギー最適化等の研究に従事．博士（工学）．IEEE 会員．

川島　裕崇（正会員）

2007 年名古屋大学大学院情報科学研究科修士課程修了．2010 年同研究科博士課程満期退学．同年名古屋大学組込みシステム研究センター研究員．算術演算アルゴリズム，組込みシステムの低消費電力化の研究に従事．修士（情報科学）．電子情報通信学会会員．

冨山　宏之（正会員）

1999 年 3 月九州大学大学院システム情報科学研究科博士後期課程修了．同年米国カリフォルニア大学アーバイン校客員研究員．2001 年（財）九州システム情報技術研究所研究員．2003 年名古屋大学大学院情報科学研究科講師，2004 年同助教授．2010 年より立命館大学理工学部教授．SOC や組込みシステムの設計技術に関する研究に従事．情報処理学会 TSLDM 編集委員長．電子情報通信学会，ACM，IEEE 各会員．博士（工学）．

高田　広章（正会員）

名古屋大学大学院情報科学研究科情報システム学専攻教授．1988 年東京大学大学院理学系研究科情報科学専攻修士課程修了．同専攻助手，豊橋技術科学大学情報工学系助教授等を経て，2003 年より現職．2006 年より大学院情報科学研究科附属組込みシステム研究センター長を兼務．リアルタイム OS，リアルタイムスケジューリング理論，組込みシステム開発技術等の研究に従事．オープンソースの ITRON 仕様 OS 等を開発する TOPPERS プロジェクトを主宰．博士（理学）．IEEE，ACM，電子情報通信学会，日本ソフトウェア科学会，自動車技術会各会員．