社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

実時間ソフトウェア再利用のためのパラメトリック実行時間解析の一手法

河井　敏宏†　　中田　明夫††

† 広島市立大学 情報科学部
†† 広島市立大学 大学院情報科学研究科

あらまし　大規模ソフトウェアの開発期間短縮には，一般にソフトウェア再利用が有効である．しかし，ソフトウェアの実行時間はプロセッサ，メモリ，OS など実行環境に依存するため，組込みソフトウェアなどの実時間ソフトウェアの再利用は一般に困難である．本研究では，そのような実時間ソフトウェアの再利用を支援するために，ソフトウェアが実時間制約を満たすために，ソフトウェアの設計パラメータおよび実行環境のパラメータが満たすべき条件を自動導出する手法を提案する．ここで，ソフトウェアの設計パラメータとしては，ソフトウェアの最低限の機能を保存しつつ実行時間を変化させるパラメータであるとし，プログラムのループ回数でパラメータ化されるものとする．実行環境パラメータとしてはプロセッサの動作周波数および各命令実行クロック数の上限・下限とする．従来のソフトウェア実行時間解析手法では，実行環境パラメータを具体的に与える必要があり，それらの最適値は試行錯誤で探す必要があったが，提案手法により，実時間制約を満たすために最適なパラメータ条件を選択することが可能となる．簡単な画像処理プログラムに対する適用例も示す．

キーワード　実時間ソフトウェア，組込みソフトウェア，実時間制約，最悪実行時間解析，形式的手法，パラメトリックモデル検査

A Parametric Execution Time Analysis Method for Reuse of Real-time Software

Toshihiro KAWAI† and Akio NAKATA††

† Faculty of Information Sciences, Hiroshima City University
†† Graduate School of Information Sciences, Hiroshima City University

Abstract　Software reuse is generally useful to reduce development time of large-scale software. However, it is very challenging to reuse real-time software such as embedded software, since the execution time of software strongly depends on its execution environment consisting of CPU, OS, memory, and so on. In this paper, to make real-time software reuse easier, we propose a method to derive automatically a condition between software design parameters and execution environment parameters in order to satisfy a given real-time constraint. Here, the software design parameters are those which changing their value does not affect the least functionality but the quality and the execution time of the software, and are related to loop iteration numbers. Execution environment parameters are CPU clock frequency and each instruction's execution cycle (including memory access time). The proposed method enables us to easily choose optimal parameter values that is ensured to guarantee the given real-time constraint without try and error.

Key words　real-time software, embedded software, real-time constraint, worst case execution time (WCET) analysis, formal method, parametric model checking

1. ま え が き

近年，組込みソフトウェアなどにおいて実時間性を求められるソフトウェア開発の需要が高まっている．一方で組込みソフトウェアの高機能化・大規模化が進んでおり，一方で市場競争により，開発期間短縮が要求される一方で，実時間制約の充足と同時に低コスト化，低消費電力化の達成が求められている．

大規模化する実時間ソフトウェアの開発期間短縮のためには，従来のソフトウェアと同様にソフトウェア部品の再利用が有効であると考えられる．しかし，特に実時間性を持つソフトウェ

ア部品に関しては，実行時間がプロセッサなどの実行環境に依存するため，異なる環境への再利用は一般に困難である．しかし，例えば実時間動画圧縮ソフトウェアや暗号処理ソフトウェアなどにおいては，圧縮率，解像度，暗号強度 (暗号化ラウンド数) など，それを変更することによってソフトウェアの機能を損なわず，処理の品質のみに影響する設計パラメータが存在する．それらのパラメータの多くはプログラムのループ回数と関連する．それらのパラメータおよび CPU の動作周波数の最小限の調整により，新しい動作環境でプログラムが時間制約を満たしかつ低消費電力で実行できるようにする最適設計支援手法が望まれる．

ソフトウェアの実時間制約の保証に関しては，従来，最悪実行時間 (Worst Case Execution Time:WCET) 解析手法 [1], [2] が研究されてきた．WCET 解析手法は基本的には，プログラムのすべての実行パス p に対して，p 上のすべてのプログラムコード断片の実行時間 $E_{p,i}$ をその断片の実行パス上における実行回数 $n_{p,i}$ で乗じたものの総和のうち最も大きい値，すなわち $WCET = \max_p\{\Sigma_i(E_{p,i} \times n_{p,i})\}$ を最悪実行時間として導出する手法である．ただし，プログラムのコード断片の実行回数 $n_{p,i}$，端的にはループ回数の上限をあらかじめ与える必要がある．なぜならばプログラムが何回ループを繰り返すかを求める問題はプログラムの停止問題に帰着され，一般に決定不能 [3] となるからである．精度のよい見積もりのために，分岐条件を考慮して実際には実行できないパスを排除したり (Infeasible Path Elimination [4])，プログラム実行を有限状態モデルで近似してループ回数の見積もりを行う研究 [5] もある．さらに正確な見積もりのために，キャッシュやパイプラインなどのプロセッサアーキテクチャをモデル化し，詳細な解析を行う研究もある [6], [7].

これら既存の WCET 解析手法を実時間ソフトウェア再利用に用いる場合，次のような問題が生じる．

- 実行時間解析の結果，実時間制約を満たさないと判明した場合に，制約を満たすようにソースコードを修正する一般的な方法がないため，試行錯誤による修正が必要である．
- 再利用対象のソフトウェアを最適に実行できる動作環境の仕様決定に用いる場合には，実行環境をいろいろ変えながら試行錯誤で実行時間解析を行う必要がある．

本研究では，これらの問題点を同時に解決するために, [8] の理論を実時間ソフトウェアに応用したパラメトリックプログラム実行時間解析手法を提案する．提案手法では，単一 CPU 単一タスクの実行環境で動作する実時間ソフトウェアを対象として，(1) 実時間プログラムのソースコード，(2) 実行環境の情報，および，(3) プログラムの実行時間に関する実時間制約が与えられたときに，実時間制約を満たしてプログラムが実行できるための，プログラムパラメータおよび実行環境パラメータに関する条件式を自動導出する．ここで，プログラムパラメータとしては，プログラムの指定した箇所のループ回数であるとし，実行環境パラメータとしては，CPU の動作周波数および各命令語の実行に要する所要クロック数の上限および下限であるとする．

提案手法の概要は次のとおりである．まず，設計者は手動でソースコードに**観測点** [8] および**マーカー** [9] を埋め込む．観測点とは，実時間制約の対象範囲の指定であり，プログラムで行われる入出力動作のポイントを，ソースコード内に明示したものである．また，マーカーとは実行回数がプログラムパラメータと関連するプログラムポイントの指定であり，本論文では，指定した点を含む最内のループの実行回数とプログラムパラメータとの関連付けをソースコード内に明示したものである．次に，コンパイル後のアセンブラコードを解析し，プログラムの機械語レベルの制御フローと各命令語の実行時間の情報を持つ状態遷移モデル (パラメトリック時間インターバルオートマトン [8]) に変換する．命令語レベルの粒度で解析することにより，コンパイラ最適化によるコード移動等に伴う実行時間の変化に対応し，より正確な実行時間見積もりが行える一方，プロセッサアーキテクチャの詳細な影響は捨象し，各命令語の実行時間の上限と下限を与えることにより，解析に要する計算量を削減する．最後に, [8] の理論に基づいて，観測点間の実行時間情報を保存した状態縮約およびパラメータ条件式導出を行うことにより，時間制約を満たしてプログラムが動作するためのプログラムの各ループ回数とプロセッサ命令語の実行時間（一般に動作周波数と命令実行サイクル数の関数）に関する条件式を導出する．

2. 問題定義

2.1 対象とする実時間ソフトウェア

本研究では，C 言語(注1)で記述された実時間ソフトウェアのプログラムソースコードを解析対象とする．ただし，解析を容易にするために再帰呼び出しおよび値にループ回数が依存するループが無いものに限定する．また，再帰的でない関数呼び出しはインライン展開されているとする．システムコール，ライブラリコールは含んでも良いが，それらの実行時間の上限・下限はパラメータとして別途与えられるものと仮定する．プログラムの各ループボディの先頭には，そのループ回数を表すパラメータ変数，または，定数がマーカーとして指定されているものとする．マーカーの指定はインラインアセンブラ構文を用いて，”marker=N”(N は変数名又は定数) という文字列をコンパイル後のアセンブラコードに埋め込むことによって行う．コンパイラに gcc を用いた場合のマーカー指定の例を図 1 に示す．

2.2 対象とする実行環境

対象とする実時間ソフトウェア実行環境としては，単一 CPU 単一タスクに限定する．CPU アーキテクチャとしては，同時に１命令のみを実行可能なシングルスカラ CPU を対象とし，各命令語の実行時間の上限と下限が，内部処理時間 (レジスタ間のみの演算など，メモリバスを使用しない処理時間)，逐次メモリアクセス時間 (連続するメモリアドレスに対するバーストアクセスの場合の 1 アドレスあたりのアクセス時間)，および，非逐次メモリアクセス時間 (非連続の独立したメモリアドレスに対するアクセス時間) の関数で表されるものとする．

命令パイプライン機構は無いもの，あるいは，レジスタ競合によるパイプラインハザードの解析が不要である３段パイプラインを持つものを対象とする．命令パイプラインがある場合

(注1)：インラインアセンブラ構文を持つプログラム言語であれば C 言語以外でも適用可能である．

```
{ ...
  for(i=0;i<N;i++} {
    /* このループを回る回数はパラメータ N で指定 */
    asm volatile ("marker=N");
    ...
    for(j=0;j<3;j++) {
      /* このループを回る回数は 3 回 (固定) */
      asm volatile ("marker=3");
      ...    }
    ...   }
  ... }
```

図 1　マーカー指定の例

Fig. 1　Example of Specifying Markers

命令語のカテゴリ	実行時間 (CPU サイクル数)
レジスタ間データ処理命令	S
メモリロード命令	$S+N+I$
メモリストア命令	$2N$
分岐命令	$2S+N$
乗算命令	$S+mI$ ($m \in \{1,\ldots,4\}$)

パラメータ:

I ：内部データ処理サイクル数 (レジスタ間転送のみ)

S ：逐次メモリアクセスサイクル数

N ：非逐次メモリアクセスサイクル数

図 2　ARM7TDMI プロセッサの命令実行時間 ([10] より抜粋)

Fig. 2　Parametric Instruction Execution Times of ARM7TDMI Processor (from [10])

は，常に次の命令語のフェッチおよびデコード（パイプラインリフィル）が終わった状態を定常状態としたときの実行時間で評価する．メモリキャッシュおよび分岐予測機構などに関しては，最良時および最悪時の実行時間で評価するものとする．

例として，ARM 社の組込み用 32bit RISC プロセッサ ARM7TDMI [10] の各命令の実行時間（一部）を図 2 に示す．ARM7TDMI プロセッサは 3 段パイプラインアーキテクチャで，メモリキャッシュを持たないため，命令実行時間の不確定性はほとんどなく，乗算命令の実行時間のみ，乗数のデータ内容に依存した不確定性が存在する．

2.3　実時間制約の指定

実時間制約とは，プログラムソースコードの指定した範囲の実行時間がある値以下（または以上）であることを要求する制約であると定義する．

実時間制約の対象範囲は，ソースコードの該当範囲の開始点および終了点に注釈を記述することにより指定する．この注釈のことを**観測点 (observation point)** と呼ぶ．本研究では，観測点の指定をインラインアセンブラ構文によって，観測点を識別する特定の文字列 (“op$n$”(n は番号)) を指定することにより記述する．観測点指定の例を図 3 に示す．実時間制約は，ある観測点 op から別の観測点 op' までの実行時間 $ET(op, op')$ に対する制約条件 $ET(op, op') \leqq T$ (又は $ET(op, op') \geqq T$ であると定義する．ここで，$ET(op, op')$ は一般にマーカーで指定したプログラムパラメータおよび動作周波数や命令実行サイクルに関するパラメータを含む．また，T は実時間制約に関するパラメータ又は定数である．

```
{ ...
  asm volatile ("op1"); /* 観測点 1 */
  ...
  for(i=0;i<N;i++} {
    ...
    for(j=0;j<3;j++) {
      ...    }
    ...   }
  asm volatile ("op2"); /* 観測点 2 */
    /* 時間制約:
       「op1 から op2 までの実行時間が T 秒以内」 */
  ... }
```

図 3　観測点指定の例

Fig. 3　Example of Specifying Observation Points

2.4　パラメトリック解析問題

実時間プログラムの**パラメトリック解析問題**とは，プログラムのソースコード $P(\vec{x})$)，実行環境 $Env(\vec{z})$，および，実時間制約 $Prop(\vec{z})$ ($\vec{x},\vec{y},\vec{z}$ はそれぞれパラメータ群) が与えられたとき，$P(\vec{x})$ が $Env(\vec{y})$ 上で $Prop(\vec{z})$ を満たして実行できるために，$\vec{x}$,$\vec{y}$,$\vec{z}$ が満たすべき条件式 $Pcond(\vec{x},\vec{y},\vec{z})$ を導出する問題であると定義する．

3.　パラメータを考慮した実時間ソフトウェアのモデル化

3.1　パラメトリック時間インターバルオートマトン

本研究では実時間ソフトウェアの動作を [8] で提案したパラメトリック時間インターバルオートマトン (以下，PTIA と呼ぶ) によってモデル化する．PTIA はパラメトリック時間オートマトン (PTA [11]) のサブクラスであり，隣接する遷移間の (パラメータ付) 時間制約を時間インターバルの形式で記述可能なモデルである．PTIA では一般に PTA よりも効率よく解析が可能である [8]．PTIA の定義を以下に再掲する．

動作の集合を Act，パラメータ変数を含む変数の集合を Var とする．$\boldsymbol{R}$ を実数全体の集合，$\boldsymbol{R}^+$ を非負実数全体の集合とする．$Intvl(Var)$ を $e_1 \leqq t$，$\leqq t \leqq e_2$，または $e_1 \leqq t \wedge t \leqq e_2$ のいずれかで表される式，およびそれらの論理結合の集合と定義する．ここで，e_1 と e_2 は $Var \setminus \{t\}$ 上の変数と R 上の定数からなる線形式 (加減算のみ用いた式) とする．直観的には t は現在の制御状態に最後に訪れてからの経過時間を表す特別な変数を表す．

［定義 1］ **パラメトリック時間インターバルオートマトン (PTIA)** とは 5 字組 $\langle S, t, PVar, E, s_{init}\rangle$ である．ただし，S は状態の有限集合，$t \subseteqq Var$ はクロック変数，$PVar \subseteqq Var$ はパラメータ変数の有限集合，$E \subseteqq S \times (Act \cup \{\tau\}) \times Intvl(PVar) \times S$ は遷移関係の有限集合，$s_{init} \in S$ は初期状態である．ここで，τ は内部動作を表す．一方，Act 上にある他の全ての変数は観測可能な動作として表現される．　□

クロック変数 t とパラメータが条件式 P を満たすとき，状態 s_i から動作 a が実行可能となり，その後状態 s_j に遷移するこ

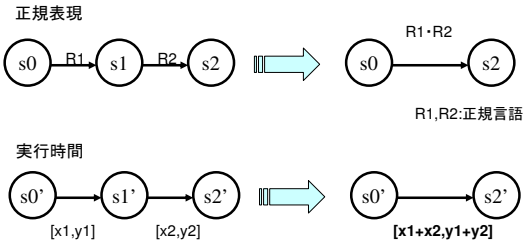

図 4　実行時間の連接演算
Fig. 4　Sequential Composition Operation of Execution Times

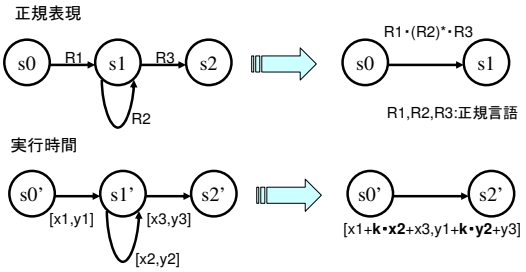

図 5　実行時間の閉包演算
Fig. 5　Kleene Closure Operation of Execution Times

とを $s{-}a[P]{\rightarrow}s'$ と略記する．状態 s_j に遷移した後，クロック変数 t は 0 にリセットされる．

PTIA のセマンティクスは PTA と同様なので定義を省略する．詳細は [8] を参照されたい．

3.2　実時間ソフトウェアと実行環境からパラメトリック時間インターバルオートマトンへの変換

実時間ソフトウェアと実行環境からパラメトリック時間インターバルオートマトンへの変換は，次のように行う．

（1） プログラムをコンパイル後のアセンブラコードから，制御構造を抽出し，プログラムアドレスを状態，命令を遷移とする有限オートマトンを構成する．この構成の際にデータ依存の制御は考慮せず，条件分岐は非決定性分岐に変換する．観測点やマーカーの指定も，通常の命令と同様に 1 遷移に変換する．

（2） 得られた有限オートマトンの各遷移に，命令実行時間の上限と下限を時間インターバルの形で付加し，PTIA を構成する．ただし，観測点やマーカーは実行時間 0 の遷移に変換する．

4.　パラメトリック解析手法

実時間ソフトウェアをモデル化した PTIA に対して, [8] の手法に基づいて次のように解析を行い，各観測点間の実行時間の上限・下限を表す式を導出する．

PTIA の各遷移のうち，観測点遷移以外の遷移を**内部遷移**と呼び，τ で表す．また，$[e_1 \leqq t \wedge t \leqq e_2]$ の形の遷移条件を $[e_1, e_2]$ と略記する．

PTIA の観測点間の実行時間条件の導出は，非決定性有限オートマトンから正規表現への変換アルゴリズム [3] と同様に行うことができる．正規表現上の演算である連接，閉包，和集合はそれぞれ図 4,5,6 のように，実行時間の構成と対応する．この事実を一般化すれば，図 7 のアルゴリズム $RIP(s_{rip})$ [8] を得る．アルゴリズム $RIP(s_{rip})$ によって，観測点間の実行時間を保存したまま，PTIA の任意の状態 s_{rip} を除去することができる．

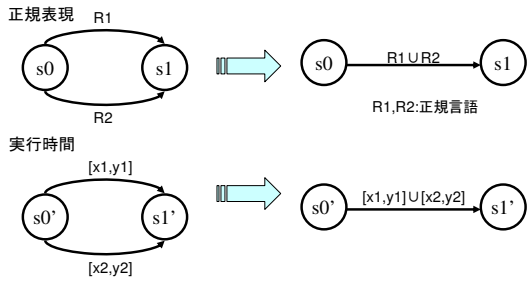

図 6　実行時間の和集合演算
Fig. 6　Union Operation of Execution Times

1: $RIP(s_{rip})$ { 実行時間を保存したまま状態 s_{rip} を除去 }
2: s_{rip} に入る内部遷移の集合 $IN(s_{rip}) \stackrel{\text{def}}{=} \{s^i_{in}{-}\tau[x^i_{in}, y^i_{in}]{\rightarrow}s_{rip} \mid i \in I, s^i_{in} \neq s_{rip}\}$ を求める
3: s_{rip} から出る任意の遷移 (観測点遷移または内部遷移) の集合 $OUT(s_{rip}) \stackrel{\text{def}}{=} \{s_{rip}{-}\alpha_j[x^j_{out}, y^j_{out}]{\rightarrow}s^j_{out} \mid j \in J, \alpha_j \in Act \cup \{\tau\}, s^j_{out} \neq s_{rip}\}$ を求める
4: s_{rip} で自己ループする内部遷移の集合 $LOOP(s_{rip}) \stackrel{\text{def}}{=} \{s_{rip}{-}\tau[x^l_{loop}, y^l_{loop}]{\rightarrow}s_{rip} \mid l \in L\}$ を求める．
5: $IN(s_{rip})$,$OUT(s_{rip})$,$LOOP(s_{rip})$ に属す遷移の内，遷移元状態と遷移先状態がそれぞれ等しい 2 つの遷移がもし存在すれば，それらの実行時間条件の和集合演算を取った遷移で置き換える．
6: もし $LOOP(s_{rip})$ の遷移中にマーカー $m_i(X)$ が存在するならば，$K \Leftarrow (X-1)$ とする．さもなければ，K は新しい変数名とする．
7: 任意の $i \in I$ と $j \in J$ の組に対して，遷移 $s^i_{in}{-}\alpha_j[\bigvee_{l\in L}\{[x^i_{in} + K * (x^l_{loop}) + x^j_{out}]\}]{\rightarrow}s^j_{out}$ を PTIA に追加する．
8: $IN(s_{rip})$,$OUT(s_{rip})$,$LOOP(s_{rip})$ に属す遷移をすべて削除する．

図 7　アルゴリズム $RIP(s)$ (文献 [8] より一部変更)
Fig. 7　Algorithm $RIP(s)$ (slightly modified from Ref. [8])

しかし，上記のアルゴリズムを本研究に適用する際に，除去する状態の順序によって一般に異なる条件式が導出されるという問題が発生する．一般に，同じ有限オートマトンを表す正規表現は複数通りあり，一意に定まらない．例えば，$a(b \cup c)^* d = a(b^* c)^* d$ が成り立つ．ここで，正規表現の連接演算が逐次実行，和集合演算が分岐，閉包演算がループにそれぞれ対応することに注意すると，左辺の正規表現はループ内部で 2 つに分岐している単純なループ構造に対応するが，右辺は全く異なるループ構造に対応することになる．

各閉包演算毎に，プログラムパラメータとは無関係のループパラメータ変数を新規に導入し，それぞれの値を調整することにより，全体の実行時間を調整するという立場 [8] であれば問題ないが，ソースコード上でそれを実際に行うのは困難である．従って，ソースコードにてマーカーで指定された特定ループの実行回数とパラメータ変数の対応関係を維持する必要がある．このためには，以下の例に示すように，適切な順序で状態除去を行わなければならない．

[例 1]　図 8 に示す PTIA を考える．遷移のうち，$m1(X)$ はマーカーの指定であり，変数 X がこの遷移を含む最内ループ回数を表すパラメータであることを意味する．$op1$ は観測点であり，状態 s_1 から観測点遷移実行後の状態 s_6 までの実行時間

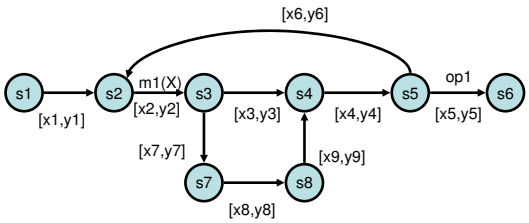

図 8　マーカー $m1(X)$ と観測点 $op1$ が指定された PTIA の例
Fig. 8　Example of PTIA w/ Marker $m1(X)$ and Observation Point $op1$

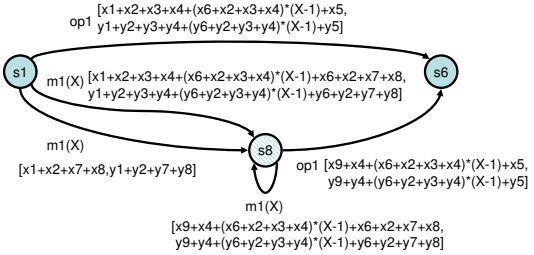

図 9　図 8 に対して誤った順序で状態除去を行った場合
Fig. 9　Case of Eliminating States of Fig. 8in Wrong Order

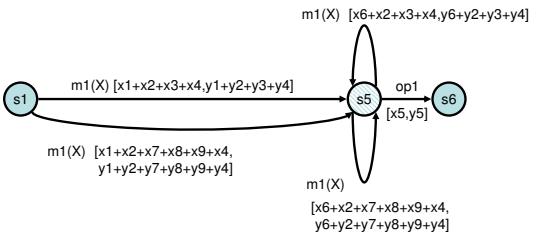

図 10　図 8 に対して正しい順序で状態除去を行った場合
Fig. 10　Case of Eliminating States of Fig. 8in Right Order

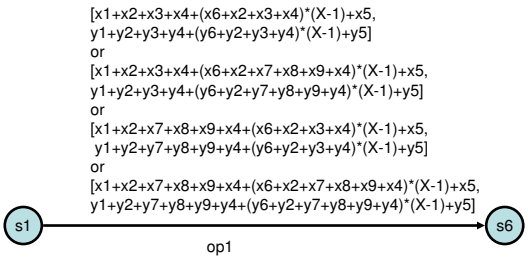

図 11　図 10 から s_5 を除去した最終結果
Fig. 11　Final Result of Eliminating s_5 from Fig. 10

を表す式を，状態 $s_2, \ldots, s_5, s_7, s_8$ を除去することによって導出することを考える．明らかに $m1(X)$ を含むループは 1 重であり，得られる式は X の一次式であることを期待する．しかし，もし $s_2, s_3, s_4, s_5, s_7, s_8$ の順で状態除去を行うと，s_7 まで除去した時点で図 9 のようになり，さらに s_8 を除去すると X の 2 次式が導出されてしまう．一方，$s_2, s_3, s_4, s_7, s_8, s_5$ の順で状態除去を行うと，s_8 まで除去した時点で図 10 のようになり，さらに s_5 を除去すると図 11 のように，ループ構造を正しく反映した条件式が導出される．　□

一般に，ループ内部において分岐が存在する場合，一方の分岐パス上の状態群のみを除去してループの縮約を行ってしまう

```
1: PETA(s) { 状態 s から到達可能なすべての内部遷移を除去 }
2: if 既に s を訪れている then
3:     return
4: else
5:     s を訪問済みとマークする
6:     I ⇐ (s から出ている内部遷移の集合)
7:     while I ≠ ∅ do
8:         各 e ∈ I に対して，もし e の遷移先状態 s' から出る遷移が 1
           以下ならば，RIP(s') を実行
9:         もし，遷移先状態 s' から出る遷移が 1 以下であるような
           e ∈ I が存在しないならば，任意の e ∈ I の遷移先状態 s' に
           対して RIP(s') を実行
10:        I ⇐ (s から出ている内部遷移の集合)
11:    end while
12:    J ⇐ (s から出ている観測点遷移の集合)
13:    for all e ∈ J do
14:        e の遷移先状態 s' に対して PETA(s') を再帰的に実行
15:    end for
16:    K ⇐ (s で自己ループしている内部遷移の集合)
17:    if K ≠ ∅ then
18:        P ⇐ (K の各要素の実行時間条件の論理和)
19:        P に ((マーカーで指定されたループ回数パラメータ) – 1) を
           乗じる
20:        J ⇐ (s から出る遷移の集合)
21:        for all e ∈ J do
22:            e の実行時間条件に P を連接
23:        end for
24:    end if
25:    return
26: end if
```

図 12　アルゴリズム $PETA(s)$
Fig. 12　Algorithm $PETA(s)$

と，ループを縮約後の実行時間条件がもう一方の分岐パス上に現れることになり，ループ回数パラメータを 2 重に乗じてしまう原因となる．ループ終了点は必ず次の繰り返しを実行するか，ループを脱出するかの分岐となっているので，この状況を回避するためには，一方の分岐パスを次の分岐点まで処理したら，もう一方の分岐パスの処理を行ってから分岐点の状態除去を行うようにすれば，ループの縮約の前にループ内部の処理を終えることを保証できる．このルールに従った順序で，初期状態 s から順に状態除去を行い，観測点遷移以外の内部遷移を除去するアルゴリズム $PETA(s)$ を図 12 に示す．アルゴリズム $PETA(s)$ の後半では，$RIP(s_{rip})$ で除去できない初期状態での自己ループ遷移の除去を行っている．

5. 実　　験

5.1 実験概要

前節のアルゴリズムを実装し，[9] で挙げられている画像処理プログラムを例題として，提案手法の適用実験を行った．この例題は，工場等の生産ラインにおいて，ラインを流れる製品の位置測定のために，製品をカメラで撮影し，得られた画像の輝度成分だけを抽出し，輝度に応じた各画素への重み付けに基づいて重心を計算し，製品の位置を推定するプログラムである．ライブラリ関数としては整数除算を使用している．動作環境と

しては，ARM7TDMI プロセッサを想定する．プロセッサのパラメータは動作周波数 f および非逐次メモリアクセスサイクル数 N とし，$f \leqq 30$Mhz の時 $N = 1$，$30 \leqq f \leqq 90$Mhz のとき $N = 2$ という制約があるとする．逐次メモリアクセスサイクル数 S および内部データ処理サイクル数 I は共に 1 で固定とする．整数除算ライブラリの実行サイクル数 div は 250 であるとする．プログラムの規模は，ソースプログラムが 87 行，コンパイル後の命令数が 256 である．プログラムは 4 重の固定回数ループで構成され，内側の 2 つのループが回数固定，外側の 2 つのループがそれぞれ画像の縦画素数，横画素数の回数分実行される．プログラムのパラメータは，画像の縦画素数 MAX_COLS および横画素数 MAX_ROWS であるとする．これに対応し，プログラム中で対応するループの先頭にマーカーを挿入し，プログラムパラメータとループ回数の対応関係を記述した．実時間制約としては，一画面分の処理を終えるまでの時間が T 秒以内，とする．これに対応し，プログラムソースコードの該当部分の前後に観測点を挿入した．解析プログラムの実行環境は，CPU が Intel Core2 Duo 2.33GHz, OS が Windows XP Professional (Cygwin 環境), メモリが 2GB である．

実験では，まず，例題から解析で得られるパラメータ条件式，および導出にかかる計算時間を評価し，次に得られた条件式を用いて，動作周波数とプログラムパラメータのトレードオフを考慮した最適化を試みる．

5.2 実験結果

実験の結果，プログラムの観測点間の最悪実行サイクル数 $WCEC$ として次の式を得た：

$$
\begin{aligned}
&I*250 + I*\texttt{MAX\_COLS}*24 + I*\texttt{MAX\_COLS}*\texttt{MAX\_ROWS}*12 \\
&+I*\texttt{MAX\_ROWS}*(-3) + \texttt{MAX\_COLS}*\texttt{MAX\_ROWS}*N*40 \\
&+\texttt{MAX\_COLS}*\texttt{MAX\_ROWS}*S*60 + \texttt{MAX\_COLS}*N*40 \\
&+\texttt{MAX\_COLS}*S*84 + \texttt{MAX\_ROWS}*N*(-10) \\
&+\texttt{MAX\_ROWS}*S*(-15) + N*443 + S*473 + div*2
\end{aligned}
$$

導出に要した CPU 時間は 0.1 秒未満であった．最悪実行時間 $WCET$ は $WCEC/f$ であるので，実時間制約を表現する式は $WCEC/f \leqq T$ となる．従って実時間制約を満たすための f に関する条件式は $WCEC/T \leqq f$ で表される．簡単のため，$\texttt{MAX\_ROWS} = \texttt{MAX\_COLS}$ とし，実時間制約が $T = 1/10$ [秒] である場合の処理画素数 MAX_ROWS と動作周波数 f の関係を図 13 に示す．図 13 において，$f \leqq 30$[MHz] の範囲では下側の曲線から $f = 30$[MHz] までの領域，$f > 30$[MHz] の範囲では上側の曲線より上の領域が，実時間制約を満たす f と MAX_ROWS の値の組の領域である．例えば，$\texttt{MAX\_ROWS} = \texttt{MAX\_COLS} = 150$ の場合は，$f \approx 26.7$[MHz] を選択するのが消費電力の観点から最適であり，$f = 40$[MHz] で動作させる場合は $\texttt{MAX\_ROWS} = \texttt{MAX\_COLS} \leqq 157$ であれば処理可能であることが分かる．

6. あとがき

本研究では，実時間組込みソフトウェアのパラメータを考慮した実行時間解析ツールを試作し，例題の画像処理プログラムに対して有効性を評価した．解析の実行時間は実用上十分高速であり，解析結果から，プログラムの処理品質と消費電力（動作周波数）のトレードオフを考慮した最適パラメータ選択が可能であることを示した．今後の課題としては，より規模の大きいプログラムでの評価，及び，関数呼び出しやマルチタスクプログラムへの拡張などが考えられる．

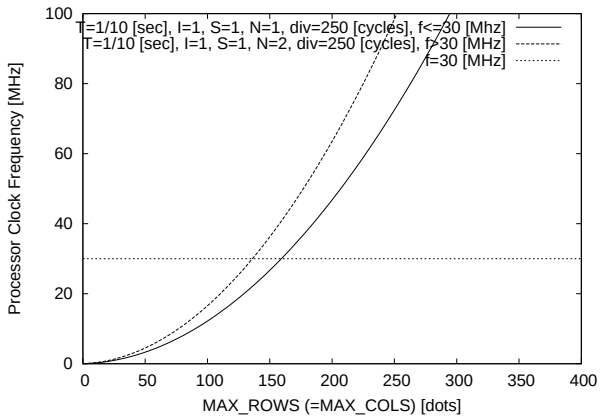

図 13 例題プログラムにおける処理画素数と最低動作周波数の関係
Fig. 13 Relation Between Resolution of Image and Minimum Clock Frequency

文献

[1] P. Puschner and A. Burns: “A review of worst-case execution-time analysis”, The Int. Journal of Time-Critical Computing Systems, **18**, pp. 115–128 (2000).

[2] J. Engblom, A. Ermedahl, M. Sjödin, J. Gustafsson and H. Hansson: “Worst-case execution-time analysis for embedded real-time systems”, Int. Journal of Software Tools for Technology Transfer, **4**, pp. 437–455 (2003).

[3] M. Sipser: “Introduction to the Theory of Computation”, PWS Publishing Company, 1st edition (1996).

[4] V. Suhendra, T. Mitra, A. Roychoudhury and T. Chen: “Efficient detection and exploitation of infeasible paths for software timing analysis”, Proc. of the 43rd ACM/IEEE Int. Design Automation Conference (DAC 2006), pp. 358–363 (2006).

[5] J. Gustafsson, A. Ermedahl, C. Sandberg and B. Lisper: “Automatic derivation of loop bounds and infeasible paths for WCET analysis using abstract execution”, Proc. of the 27th Int. Real Time Systems Symposium (RTSS 2006), IEEE Computer Society Press (2006).

[6] A. Metzner: “Why model checking can improve WCET analysis”, Proc. of the 16th Int. Conf. on Computer Aided Verification (CAV 2004) (Eds. by R. Alur and D. Peled), Vol. 3114 of Lecture Notes in Computer Science, Springer, pp. 334–347 (2004).

[7] J. Engblom: “Processor Pipelines and Static Worst-Case Execution Time Analysis”, Upssala dissertations from faculty of science and technology 36, Uppsala University (2002).

[8] T. Tanimoto, A. Nakata, H. Hashimoto and T. Higashino: “Double depth first search based parametric analysis for parametric time-interval automata”, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, **E88-A**, 11, pp. 3007–3021 (2005).

[9] P. Puschner and C. Koza: “Calculating the maximum, execution time of real-time programs”, Real-Time Systems, **1**, 2, pp. 159–176 (1989).

[10] ARM Ltd.: “ARM7TDMI Technical Reference Manual, Revision: r4p1” (2004). http://infocenter.arm.com/help/topic/com.arm.doc.ddi0210c/.

[11] R. Alur, T. A. Henzinger and M. Y. Vardi: “Parametric real-time reasoning”, Proc. 25th ACM Annual Symp. on the Theory of Computing (STOC’93), pp. 592–601 (1993).