Vol.2010-SLDM-144 No.36
Vol.2010-EMB-16 No.36
Vol.2010-MBL-53 No.36
Vol.2010-UBI-25 No.36
2010/3/27

組込みソフトウェアにおける実行トレースに基づく捨象モデリングツールの実装と評価

高　鏘守[†1]　久住憲嗣[†2]　小野康一[†3]
河原亮[†3]　坂本佳史[†4]　中田武男[†3]
長野正[†5]　福田晃[†6]

組込みシステムでは，ユーザニーズの多様化，製品サイクルの短期化により開発期間を短くする必要がある．したがって，手戻りによるリスクを回避・軽減するため，開発早期にシステムレベルでの性能を評価する必要がある．

本研究では，組込みソフトウェアにおける実行トレースの捨象手法を提案し，実装・評価を行う．実際の MFP(Multifunction Peripheral/Printer) 製品の実行トレースに 2 つの提案手法を適用し，評価を行った．捨象前の実行トレースの量と捨象後の実行トレースの量との割合を捨象率と定義し，捨象による実行トレースの簡略化の尺度とする．20 個のテストパターンにおいて提案手法を適用し，捨象率を調べた．手法 1（呼出元と呼出先の関係による捨象）では，捨象の閾値を 90%とし捨象を行ったとき，捨象率は約 60%だった．手法 2（同一モジュール内の関数の統合による捨象）では，捨象率は約 50%だった．20 個のパターンのうち 2 つパターンでは特徴（テストパターンごとの実行トレースを識別するための固有の関数）が失われたが，それ以外のテストパターンにおいてはテストパターンの特徴を維持していた．よって，同一モジュール内の関数の統合による捨象が有効であることを示した．

Implementation and Evaluation of Eliminating Execution Traces in Embedded Systems

Yongsoo Ko,[†1] Kenji Hisazumi,[†2] Kouichi Ono,[†3]
Ryo Kawahara,[†3] Yoshifumi Sakamoto,[†4]
Takeo Nakada,[†3] Tadashi Nagano[†5]
and Akira Fukuda[†6]

We propose the method that eliminating execution traces and develope and evaluate the method in embedded systems. For evaluating proposed method, we applied to the execution trace of MFP(Multifunction Peripheral/Printer). The eliminating rate is defined as the ratio of execution trace lines before eliminating to execution trace lines after eliminating. We applied proposed method to the 20 test patterns, and showed it was effectiveness.

1. はじめに

組込みシステムに対するユーザニーズが多様化，製品サイクルの短期化により，高品質な製品を短期に開発することが求められている[1)]．しかし，性能要件が満たされないことで手戻りが発生した場合，開発時間とコストが増大することになる．このようなリスクを回避・軽減するためには，開発早期にシステムレベルでの性能を評価することが有効である．

文献 2) では，レガシーソフトウェアのマルチコア化を目的として，UML による性能評価モデルをリバースモデリング技術を用いて作成する手法を提案している．レガシーソフトウェアは長期間による差分開発により，開発当初の仕様・設計文書などが残っていないことが多い．そのため，本研究では実行トレースをもとにリバースモデリングして性能評価モデルを作成している．実行トレースは情報量が多くそのままでは性能評価モデルとして利用しにくいため，実行トレースを捨象して振舞いを抽出する．実行トレースの捨象結果から現行システムの振舞いを抽象的に表現しているモデルを作成し，それを元に次期システムの要件に基づいてモデルを変更する．その後，変更したモデルをトレース駆動シミュレーション[3),4)] によって次期システムアーキテクチャの性能評価を行っている．しかし，文献 2) は捨象が満たすべき要件は述べているが，実装については言及しておらず．100 千行を超える

†1 九州大学大学院システム情報科学府
Graduate School of Information Science and Electrical Engineering, Kyushu University
†2 九州大学 システム LSI 研究センター
System LSI Research Center, Kyushu University
†3 日本アイ・ビー・エム株式会社 東京基礎研究所
IBM Research - Tokyo
†4 日本アイ・ビー・エム株式会社 グローバルビジネスサービス 組込みソフトウェア・コンサルティング
Global Business Services, IBM Japan, Ltd.
†5 日本アイ・ビー・エム株式会社 R&D イノベーション・サービス
R&D Inovation Services, IBM Japan, Ltd.
†6 九州大学大学院システム情報科学研究院
Faculty of Information Science and Electrical Engineering, Kyushu University

Vol.2010-SLDM-144 No.36
Vol.2010-EMB-16 No.36
Vol.2010-MBL-53 No.36
Vol.2010-UBI-25 No.36
2010/3/27

ような大規模の実行トレースの捨象は考慮していない．

それを解決するため，本研究では実行トレースからのリバースモデリングのための捨象手法を提案し，評価する．実行トレースの量は組込みソフトウェアによっては膨大になるため，捨象を人手で作業するのは多大な工数を要する．そのため，提案手法では捨象を自動的に行うツールを実装する．提案手法では，振舞いの理解や，大まかなアーキテクチャ分析/性能評価に利用することができるように捨象する．

本論文の構成は以下の通りである．まず，2 章で性能評価モデリング技術について紹介し，本研究との関連性を述べる．3 章では実行トレースの捨象手法を提案する．4 章において実際の MFP(multifunction peripheral/printer) 製品に本手法を適用したケーススタディについて述べ，提案手法による有効性について実験・評価をおこなう．最後に 5 章で本研究のまとめと今後の課題を述べる．

2. 性能評価モデリング技術

文献 2) では，システムレベルのアーキテクチャ分析のためにモデルをアーキテクチャモデルと定義し，そのモデルは実行トレースからリバースモデリング手法を用いて作成する．このモデルはシステムレベルのモデルであるので高い抽象度で表現されないといけないが，そのためには捨象が必要である．システムアーキテクチャの性能を評価することを意図しているため，捨象の視点としては性能が設定されている．性能に及ぼす影響が大きい要素を残し，それ以外はその要素に統合することで，モデルの構成要素を簡潔にしている．性能に関する情報はプログラムを実行せずに得ることはできないため，システムの振舞いを表す実行トレースを性能視点で捨象して解析する．

性能評価モデリング手法の手順は以下の通りである．

(1) システム観測技術[6] を使用して対象の組込みシステムの実行トレースを取得する．
(2) 実行トレースを性能視点で捨象してアーキテクチャにおける振舞いモデルを作成する．
(3) モデルを次期製品の要件に沿って変更し，このモデリングとは独立に，関数処理時間やリソース消費量などの情報を性能指標として格納する．
(4) 変更したモデルを性能指標とともに実行することで，次期製品のシステムアーキテクチャに関する性能評価結果を得る

文献 2) では，実際の MFP 製品に手法を適用しており，MFP の主要な機能である印刷ジョブ処理に対する適用結果について述べている．現行 MFP 製品の実行トレースをシステム観測技術[6] を使用して取得している．実行においては，JEITA(電子情報技術産業協会) によるプリンタベンチマーク用のテストパターンの J12 セット[7] をテストケースとして使用している．J12 セットには 20 個のテストパターンが定義されている．J12 セットの一つのテストパターンについて MFP 製品を実行した際に取得した実行トレースの関数の呼出関係を抽出して，関数コールツリーを得ると同時に，実行トレースにおける関数呼出の性能パラメーターを抽出している．関数コールツリーを捨象した結果，ノード数は 15,340 から 155 となり，ほぼ 1/100 になっている．

評価として，実際に MFP プロトタイプを製作し，モデルによる性能評価結果が MFP プロトタイプの性能とほぼ合致することを確認し，手法による性能評価が妥当性を持つことを示している．ほとんどのテストケースについて，モデルの実行時間は MFP プロトタイプの実行時間とほぼ一致していた．

3. 提 案 手 法

文献 2) の性能評価モデリング手法の特徴的な技術は，実行トレースの捨象手法である．しかし，文献 2) は捨象が満たすべき要件は述べているが，実装については言及していない．そこで，この章では，実行トレースの捨象手法を提案する．

本研究では，では下記の 2 種類の捨象手法を提案する．

手法 1 呼出元と呼出先の関係による捨象

手法 2 同一モジュール内の関数の統合による捨象

ここで，モジュールは，ソースコード中の関数やクラスをグループ化したものである．個々の関数は，いずれかのモジュールに属する．「同一モジュール内の関数を統合する」とは，代表となる関数だけを残して，その関数は他の関数の処理も代行しているとみなすことである．具体的には，同一モジュールに属する関数の重みを最上階呼出先に集めて，一つに統合することである．

以降，捨象を適用するために満たさなければならない事前条件と提案手法の詳細について述べる．

3.1 事 前 条 件

まず，実行トレースをコールツリーに変換してから，生成されたコールツリーとモジュール分け情報から捨象を行う．この節では，実行トレース，コールツリー，モジュール分けが満たすべき条件について述べる．

3.1.1 実行トレース

今回の提案手法では，実行時間の情報を用いて性能視点で捨象する．そのため，実行ト

Vol.2010-SLDM-144 No.36
Vol.2010-EMB-16 No.36
Vol.2010-MBL-53 No.36
Vol.2010-UBI-25 No.36
2010/3/27

レースには少なくとも下記の情報が記録されていなければならない．

- 関数名
- 関数の total 時間
- 関数の self 時間
- 関数の呼出階層

total 時間と self 時間は以下のように定義する．

関数の total 時間

関数が呼ばれてから終わるまでの実行時間，つまりその関数の処理時間である

関数の self 時間

関数の total 時間から，その関数から呼ばれる全ての関数の total 時間を引いた時間である

実行トレースをコールツリーに変換するため，関数の呼出階層も記録されていなければならない．図 1 に実行トレースの例を示し，説明する．

各行は，

関数名 (self 時間 / total 時間)

で構成され，インデントにより呼出階層を表している．図 1 の funcA は Main から呼び出され，funcC と funcD を呼び出している．funcA の total 時間は 4000 であり，funcA から呼び出される全ての関数（funcC，funcD）の total 時間の合計は 2500 (1300 + 1200) である．funcA の self 時間は 1500 (4000 − 2500) である．

```
Main (3000 / 10000)
    funcA (1500 / 4000)
        funcC (1300 / 1300)
        funcD (1200 / 1200)
    funcB (1700 / 3000)
        funcE (800 / 1300)
            funcF (500 / 500)
```

図 1 実行トレースの例

3.1.2 コールツリー

コールツリーは，サブルーチン同士の呼び出し関係を表現した有向グラフである．具体的には、各ノードが関数を表現し、各エッジ (f, g) は関数 f が関数 g を呼び出すことを示す。

表 1 モジュールの関数の対応表

モジュール名	属する関数名
モジュール1	Main, funcA
モジュール2	funcB, funcC, funcE
モジュール3	funcD
モジュール4	funcF

コールツリーは実行トレースから容易に生成可能である．

図 2 は図 1 の実行トレースをコールツリーに変換したものである．

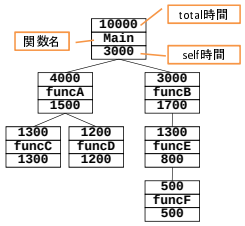

図 2 コールツリーの例

3.1.3 モジュール分け

捨象を行う前に，関数群を性能評価に必要な粒度のグループに分ける必要がある．そのグループのことを，本研究ではモジュールとする．モジュールを分ける際は，設計文書中で定義されている機能単位で分けることが望ましい．

以降のコールツリーは，モジュール分けを反映し，図示する．同じモジュールに属するノードは同じ色で表す．モジュール分けが表 1 のとき，図 2 のコールツリーは図 3 になる．

3.2 手法 1：呼出元と呼出先の関係による捨象

手法 1 では同一モジュール内の関数を統合せずに，呼出元と呼出先の関係による捨象を行う．捨象はメイン関数から開始し，全関数に対して行う．関数の選択基準は，選択した呼出先の関数群の total 時間の総和が呼出元の total 時間に対して大半を占めるようにすることである．

しかし，上記の方法では個々の関数が属するモジュールを考慮しないものになる．そこで，モジュール間のつながりを考慮した上でノードを選択する手順について次頁以降で述べる．

Vol.2010-SLDM-144 No.36
Vol.2010-EMB-16 No.36
Vol.2010-MBL-53 No.36
Vol.2010-UBI-25 No.36
2010/3/27

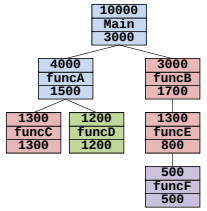

図 3 モジュール分けを反映したコールツリーの例

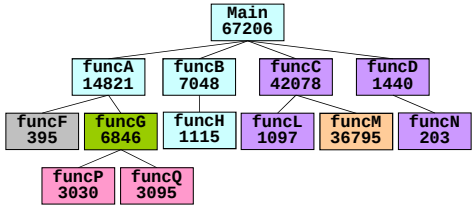

図 4 捨象前のコールツリー

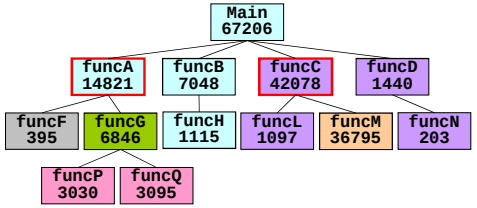

図 5 捨象中のコールツリー

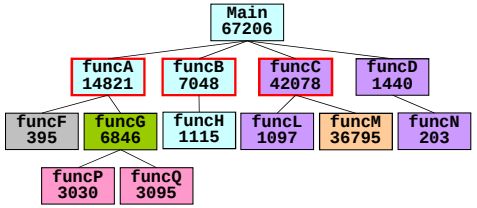

図 6 捨象中のコールツリー

捨象において，ノード選択の基準は下記の優先度で決める．

(1)　中間ノードを見つける．

(2)　それ以外のノードから時間の大きい順でノードを調べる．

(3)　1 と 2 において，閾値に達したら，終了する．

ここで，**中間ノード**とは，子孫ノードの中に自分と違うモジュールに属するノードが存在するノードである．中間ノードとなる関数は，違うモジュール同士をつないでいることになる．そのため，中間ノードを優先的に探索し，残すようにしている．

図 4，図 5，図 6，図 7 に，閾値を 90%とし，捨象を行うときの様子を図で説明する．この節で例として使うコールツリーはノードを簡略化するため，self 時間を省略し，関数名と total 時間だけを残している．図 4 は，捨象が行われる前のコールツリーである．

root となるノードから捨象が行われる．Main の self 時間は Main の total 時間から子ノードの total 時間の総和を引くことで求めることができる．中間ノードを調べる．funcA の子孫ノードに，funcA と違うモジュールに属する funcG，funcP，funcQ が存在するので，funcA は中間ノードである．同じく funcC は子孫ノードに funcM を持ち，funcM は funcC と違うモジュールに属するので，funcC も中間ノードである．よって，funcA と funcC が選択される．

次に，funcB，funcD の順 (時間の大きい順) で調べていく．まず，funcB を調べる．funcB を追加すると，今まで選択されたノードの total 時間の総和が 66396 になり，Main の 90%の時間異常になったため，終了する．funcD は選択されない．

その後，funcA，funcB，funcC の子ノードに対して捨象が行われる．

すべての手続きが終了した後，最終的に集合 U に含まれているノードが捨象によって残されたノードである．図 7 に集合 U に含まれているノードを太線で囲み，図示する．今回の例では，捨象の結果，ノード数は 13 から 8 になった（図 8）．

3.3 手法 2：同一モジュール内の関数の統合による捨象

手法 2 では同一モジュール内の関数を統合する．同一モジュールに属する関数の重みを同一モジュール内の最上階呼出先に集めて，一つにすることである．統合方法として，コール

Vol.2010-SLDM-144 No.36
Vol.2010-EMB-16 No.36
Vol.2010-MBL-53 No.36
Vol.2010-UBI-25 No.36
2010/3/27

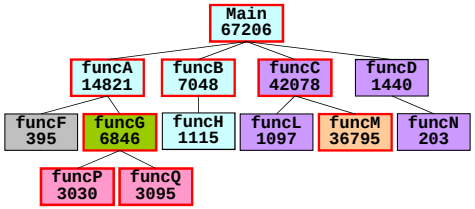

図 7 捨象の結果，残ったノード

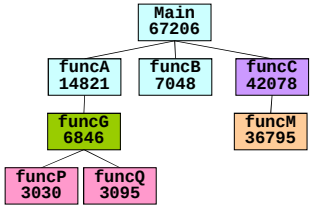

図 8 捨象後のコールツリー

ツリーの *root* ノードから再帰的に実行する．

まず，子ノードから同じモジュールに属するノードを調べる．同じモジュールに属するノードを見つけたら，そのノードを親ノードに統合する．統合されたノードは，その親ノードが処理を代行しているとみなす．そのため，統合されるノードの total 時間をその親ノードの self 時間に加算する．このことによって式**??**を満たすことができる．統合方法は，統合されるノードが中間ノードか否かによって異なる．

まず，統合されるノードが中間ノードでない場合（図 9）について述べる．

nodeA，nodeB···nodeD は同じモジュールに属しているため，最上位ノードである nodeA に nodeB···nodeD を統合することができる．nodeB の全ての子孫ノードが nodeA に統合される．そのとき，nodeA の self 時間に nodeB の total 時間を加算することで，式**??**を満たすことができる．

次に，統合されるノードが中間ノードである場合（図 10）について述べる．

nodeA，nodeB···nodeD は同じモジュールに属しているが，nodeB の子孫ノードには，

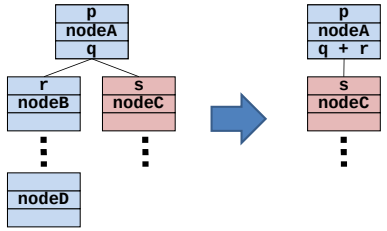

図 9 中間ノードでない場合

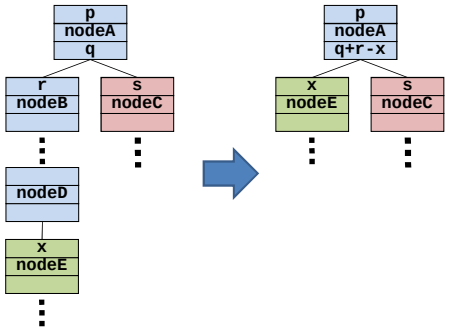

図 10 中間ノードである場合

Vol.2010-SLDM-144 No.36
Vol.2010-EMB-16 No.36
Vol.2010-MBL-53 No.36
Vol.2010-UBI-25 No.36
2010/3/27

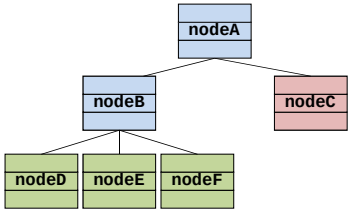

図 11 統合前のコールツリー

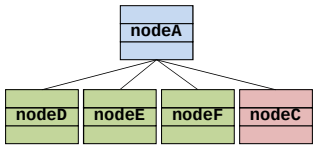

図 12 統合後のコールツリー

異なるモジュールに属するノードが存在する（nodeE）．そのため，異なるモジュールに属するノードを残しながら，統合を行う必要がある．まず，図 9 と同様に nodeB･･･nodeD を nodeA に統合する．その後 nodeE を直接 nodeA につなぐ．そのとき，nodeA の self 時間に nodeB の total 時間を加算した後，統合されないノード（nodeE）の total 時間を引くことで，式**??**を満たすことができる．図 11 のコールツリーは，統合を行うと，図 12 になる．nodeA の子ノードの数は 2 から 4 に増えた．このように，子ノードの数が増えることもある．

4. 評　　価

この章では，実際の MFP 製品の実行トレースに捨象手法を適用する．適用対象となる実行トレースは JEITA（電子情報技術産業協会）によるプリンターベンチマーク用のテストパターンの J12 セット[?)] をテストケースとして取得した．J12 セットには 20 個のテストパターンが定義されており，本研究では，すべてのテストパターンの実行トレースに対して捨象を行った．実行トレースに現れる関数の数は 212〜251，行数は約 12 千行〜約 178 千行であった．

評価の指標は以下の通りである．

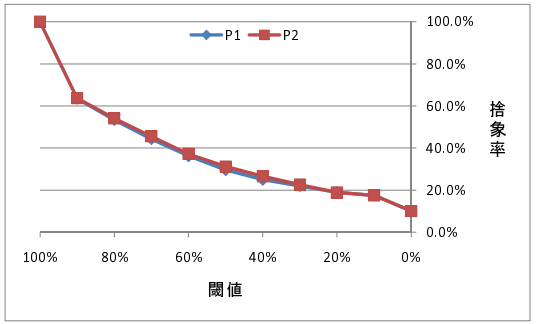

図 13 グループ 1 の捨象結果

- どのぐらいのノード数が捨象されたか
  捨象結果をもとにモデルを作成するので，ノード数は少ない方が望ましい
- 捨象後にもテストパターンの特徴を残しているか
  捨象によりテストパターン毎の違いがなくなってはならない

捨象の尺度として，**捨象率**を定義する．**捨象率**は，捨象前のノード数と捨象後のノード数の割合であるとする．

モジュール分け

本手法は，モジュール分けに依存する点が多い．手法 1 においては，中間ノードか否かを判別するためモジュール分け情報が用いられる．手法 2 では，同一モジュール内の関数を統合しているので，モジュール分けによる影響はさらに大きい．

今回は，MFP の技術文書と実行トレースの計測ポイントをもとに，モジュール分けを行った．その結果，7 つのモジュールに分けることができた．以降の手法では，この 7 つのモジュールを用いて捨象した．

4.1 手法 1（呼出元と呼出先の関係による捨象）の評価

前節のモジュール分けの結果を用いて，以下の 7 つのグループに分け，捨象結果を述べる．

テストパターン 1（P1），テストパターン 2（P2）の結果グラフ（図 13）は非常に波形が類似しており，閾値が 90%になるところまでは，急激にノード数が減っており，その後のノード数は緩やかに減少している．閾値が 90%のとき，捨象率は約 60%であった．他のパターンにおいても同様の結果が得られた．

Vol.2010-SLDM-144 No.36
Vol.2010-EMB-16 No.36
Vol.2010-MBL-53 No.36
Vol.2010-UBI-25 No.36
2010/3/27

4.2 手法 2（同一モジュール内の関数の統合による捨象）の評価

20 個のテストパターンから，実行トレースに登場する関数の種類と個数が同じであるテストパターンを同一グループとみなし，グループ分けを行った．

捨象前と捨象後で，パターンの所属するグループの変化を図 2 に示す．

表 2 捨象前と捨象後のグループの変化

捨象前	J12P1	J12P2	J12P3	J12P4	J12P5	J12P6	J12P7	J12P8	J12P9	J12P10	J12P11	J12P12	J12P13	J12P14	J12P15	J12P16	J12P17	J12P18	J12P19	J12P20
捨象後	J12P1	J12P2	J12P3	J12P4	J12P5	J12P6	J12P7	J12P8	J12P9	J12P10	J12P11	J12P12	J12P13	J12P14	J12P15	J12P16	J12P17	J12P18	J12P19	J12P20

統合前と統合後で，パターン 9 とパターン 11，12 の所属グループが変化した以外，他のグループの変化は見られない．したがって，今回のモジュール分けは概ね妥当であると考えられる．

モジュールを統合前と統合後を比較した結果について述べる．図 14 は統合前と統合後のノード数の変化をパターン別に調べたものである．図 15 はノード数の変化の割合を示している．モジュールの統合により，約 50%のノードが捨象された．

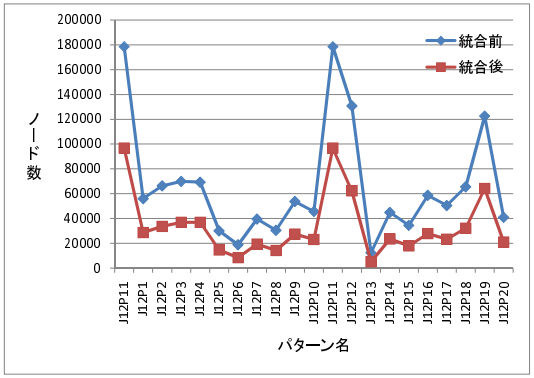

図 14 モジュール統合前と統合後のノード数

5. おわりに

本章ではまとめと今後の課題について述べる．

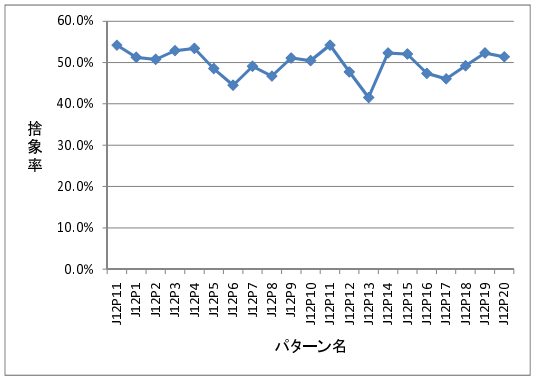

図 15 モジュール統合前と統合後のノード数

5.1 まとめ

本研究では，性能評価モデリングのための実行トレースの捨象手法を提案した．捨象を行う際に，捨象前に同一モジュール内の関数を統合するか否かで 2 つの手法を提案した．

提案手法を評価するために，実際の MFP(multifunction peripheral/printer) 製品の実行トレースに本手法を適用した．実行トレースは，JEITA（電子情報技術産業協会）によるプリンターベンチマーク用のテストパターンの J12 セット 20 個のテストパターンををテストケースとして取得した．

手法 1（モジュール統合による捨象）では，閾値を 90%として捨象を行ったとき，実行トレースは最初の約 60%になった．手法 2（モジュール統合による捨象）では，捨象前にモジュールの統合を行う．統合により実行トレースは約 50%になった．手法 2 では捨象により，20 個のパターンのうち 2 つパターンの特徴が失われたが，それ以外のテストパターンにおいては実行トレースの特徴を維持していた．よって，同一モジュール内の関数の統合による捨象が有効であることを示した．

5.2 今後の課題

本研究では，捨象を行う際，親ノードの子ノードの関係だけに着目している．呼出階層が深くなればなるほど，出現する total 時間は小さくなるため，深い呼出の場合は捨象の効果が少ない．

今後は実行トレース全体の時間の個々の関数の時間の関係に着目した捨象手法を提案し，その評価を行う予定である．

Vol.2010-SLDM-144 No.36
Vol.2010-EMB-16 No.36
Vol.2010-MBL-53 No.36
Vol.2010-UBI-25 No.36
2010/3/27

謝辞 実行トレースを提供して頂いた京セラミタ株式会社 東京 R&D センターの福岡 直明氏に謹んで感謝の意を表する．

参 考 文 献

1) 経済産業省：2009 年版組込みソフトウェア産業実態調査報告書, http://www.meti.go.jp/policy/mono_info_service/joho/downloadfiles/2009software_research/index.htm (2009)
2) 小野康一，豊田学，河原亮，坂本佳史，中田武男，福岡直明：組込みソフトウェアの性能解析のための実行トレースの捨象にもとづくモデリング手法，第 16 回ソフトウェア工学の基礎ワークショップ (FOSE 2010)，pp.1–12 (2009).
3) Jiun-Ming Hsu and Prithviraj Banerjee : “Performance measurement and trace driven simulation of parallel CAD and numeric applications on a hypercube multicomputer”, *IEEE Transactions on Parallel and Distributed Systems*, Vol.3, No.4, pp.451–464 (1992).
4) Cosimo Antonio Prete, Gianpaolo Prina, and Luigi Ricciardi : “A trace-driven simulator for performance evaluation of cache-based multiprocessor systems”, *IEEE Transactions on Parallel and Distributed Systems*, Vol.6, No.9, pp.915–929 (1995).
5) SESSAME WG2 :『組込みソフトウェア開発のためのリバースモデリング：既存のソースコードを活用したリバース設計による効率的で高品質なソフトウェア開発』, 翔泳社 (2007).
6) Nobuyuki Ohba and Kohji Takano : “Hardware debugging method based on signal transitions and transactions”, In *Proceedings of the 11th Asia South Pacific Design Automation Conference (ASP-DAC 2006)*, pp.454–459 (2006).
7) Japan Electronics & Information Technology Industries Association (JEITA) : *JEITA Printer Benchmark Test Patterns*, http://it.jeita.or.jp/document/printer/pattern/J1-J12.pdf.