Vol.2016-ARC-218 No.18
Vol.2016-SLDM174 No.18
2016/1/20

ソフトウェアオブジェクトのプロファイリング技術の開発

岩河秀知[†1]　　豊永雅彦[†1]　　村岡道明[†1]

[†1] 高知大学大学院　理学専攻　情報科学分野
〒780-8520 高知県高知市曙町 2-5-1
{hiwagawa, toyonaga, muraoka}@is.kochi-u.ac.jp

近年，組込みソフトウェアが多数の分野で利用されている．しかしながら，これらのソフトウェアの内部や性能および再利用性などについてはその詳細が明示されていないものも多く，再利用できるかどうかの判断が難しい．現在、C 言語で記述されたソフトウェアの最適化のためのソフトウェアプロファイリング技術が開発され、一部のソフトウェアの性能解析に利用されている。しかしながら、オブジェクトコードしか存在しないプログラムについてはその構造や性能の解析技術は未確立である。本稿ではオブジェクトコードの構造や性能の解析手法を提案し、それに基づく解析ツールの試作を行ったので報告する。本ツールはオブジェクトコードの構造および基本ブロックやサブルーチンの呼び出し回数や実行時間などを解析（以上をプロファイリングと呼ぶ）し、性能に関するボトルネックの検出を行う。本技術の確立によりソフトウェアの再利用性の向上を図ることが可能であることを示す。

A Development of Profiling technology for software objects

HIDETOMO IWAGAWA　MASAHIKO TOYONAGA
MICHIAKI MURAOKA

Graduate School of Science, Kochi University
2-5-1 Akebono-cho, Kochi, 780-8520, Japan

In these years, embedded software has been used in various areas. However, neither the performance, power dissipation nor the reuse rates are not known, so the reuse of the softwares are difficult. Software profiling technology for optimization of software written in C language is developed at present, and it's used for a performance analysis of software in a few cases. However, analysis technology on the structure and performance for object codes without its source code has not yet be established. An analysis method of the structure and the performance of the object code has been proposed, and made an analysis tool prototype based on the proposal. And, they are reported. The structure of the object code, a number of calls of the basic blocks and the subroutine and an execution time are analyzed by this tool, and a bottleneck of the performance is detected. The establishment of this technology shows that the improvement of reusability of software objects can be increased, respectively.

1. はじめに

現在，ソフトウェアの最適化を行うため，C 言語で記述されたソフトウェアのプロファイリング技術が開発されている．しかしながら，これはCコードで記述されたプログラムには適応できるが，オブジェクトコードしか存在しないプログラムについては，その構造や性能の解析技術は未確立である．本稿では，オブジェクトコードの構造や性能の解析手法を提案し，それに基づく解析ツールの試作を行ったので報告する．

現在開発されているC言語で記述されたソフトウェアプロファイリングの技術では，対象プロセッサ上でCコードを実行したときの実行時間をシミュレーションにより求める機能や，サブルーチンのコール回数，実行サイクル数などを確認することが可能であり，性能解析に利用されている．

本手法では，オブジェクトコードの構造および基本ブロックやサブルーチンの動作回数や実行時間などを解析（以降プロファイリングと呼ぶ）し，性能に関するボトルネックの検出を行う．本技術の確立によりソフトウェアの再利用性の向上を図ることが可能であることを示す．

2. 関連研究

現在，C 言語で記述されたソフトウェアのプロファイリングを行う Visual Spec[1][2]というプロファイリングツールがすでに開発されている．Visual SpecはC言語をファンクション単位で解析し，対象プロセッサ上での実行時間をシミュレーションを用いて導出でき，各ファンクションにおける詳細実行サイクル数や，全体サイクル数からの占有率などを確認できるツールである．Visual Spec で確認できる主要項目を以下に示す．

① ファンクションあたりの実行サイクル数
② 全体サイクル数からの占有率
③ コール回数,
④ 1ファンクションあたりの最大サイクル数
⑤ 1ファンクションあたりの最小サイクル数

これらをプロファイリング結果として出力し，プログラム中のボトルネックの検出に利用されている．

また，オブジェクトコードをアセンブリ言語やC言語へ変換することのできるIDA[3]というツールも開発されている．これは，バイナリからのアセンブルとプログラム構造の復元が行え，プログラムの呼び出し構造のグラフ表示や，ファンクションコール中のアセンブリコードの確認が行える．

本組込みコード解析ツールは，静的解析として基本ブロック単位でプログラムの構造をフローチャートで表示

ⓒ2016 Information Processing Society of Japan

する機能を作成し，動的解析として，Visual Spec の出力項目をサポートするとともに，基本ブロックごとのプロファイリング結果の出力もサポートする．

3. ソフトウェアオブジェクトのプロファイリング手法の機能仕様

本組込みコード解析手法はターゲットプロセッサのオブジェクトコードを解析する手法である．今回対象とする ARM7TDMI は，一般的に広く知られているプロセッサであるため，今回対象プロセッサとして採用している．

本組込みコード解析手法の全体構成として，大きく 2 つの解析部から構成されている．1 つはプログラム構造(制御とデータのフロー，階層)を可視化するためにフローチャートを生成する静的解析部であり，もう 1 つはオブジェクトコードを命令セットシミュレータを用いて実行し，解析結果を出力する動的解析部である．

静的解析部では，オブジェクトコードの構造を理解するために，オブジェクトコードからフローチャートを生成し，プログラムの流れを確認できる機能を付加する．

動的解析部では，オブジェクトコードを基本ブロックに分割し，基本ブロック単位で解析を行う．プロファイリングにより確認できる項目は Visual Spec で確認することができる項目を基本ブロック単位で出力する．

3.1 解析部の機能仕様

本手法は静的解析部と，動的解析部により構成されている．各解析部における機能仕様を以下に示す．

3.1.1 静的解析部機能仕様

静的解析部は, ARM7TDMI のオブジェクトコードを読み込み，フローチャートを生成することが可能である．以下に入力仕様，解析仕様，フロー生成仕様，出力仕様を示す．

a. 入力仕様

入力方式はファイル入力である．テキストファイルに 32 ビットの ARM 命令を 16 進数表記で記述したものを入力ファイルとする．

b. 解析仕様

静的部分の解析は 3 ステップに分かれてオブジェクトコードから制御およびデータフローを解析し，フローチャートなどを生成する．各ステップを以下に示す．

- 命令解析
- 基本ブロック分割
- フロー生成

各項目の詳細はフロー生成アルゴリズムで述べる．

c. 出力仕様

静的解析を行った結果をフローチャートを例として示す. フローチャートは基本ブロックを単位として生成し，フロー間の接続関係はフロー間を矢印で接続する．出力項目について以下に示す．

① フローチャート記号

フローチャートの種類を示すフローチャート記号とその意味を示すテキストを出力する．フローチャートの種類を以下に示す．

- Start：プログラムの開始
- END：プログラムの終了
- Decision：分岐処理
- While－Loop：ループ条件
- Process：一般処理
- Change-Page：ページ跨ぎ
- IF：IF を構成する基本ブロック一覧
- BLOCK：While-Loop を構成する基本ブロック一覧

② コンディション

条件分岐をする際にのみ表示．分岐するための条件を示す．コンディションが成立している場合にのみ，分岐を行う．

③ マシンコード・命令ニーモニック

基本ブロックを構成しているマシンコード・命令ニーモニックを表示するエリア．Start や END のようにコードを持たないフローにはマシンコード・命令ニーモニックエリアは付加しない．また，通常時はこのコードは隠蔽されているが，フローチャートをクリックすることで内容が確認できる仕様である．

本手法の静的解析部の実行例として，図 1 のフローチャートと同じ構造になるような C コードを作成し, ARM のオブジェクトコードにコンパイルした後に，本ツールーの静的解析を用いてフローチャートを生成すると図 2 のようになる．

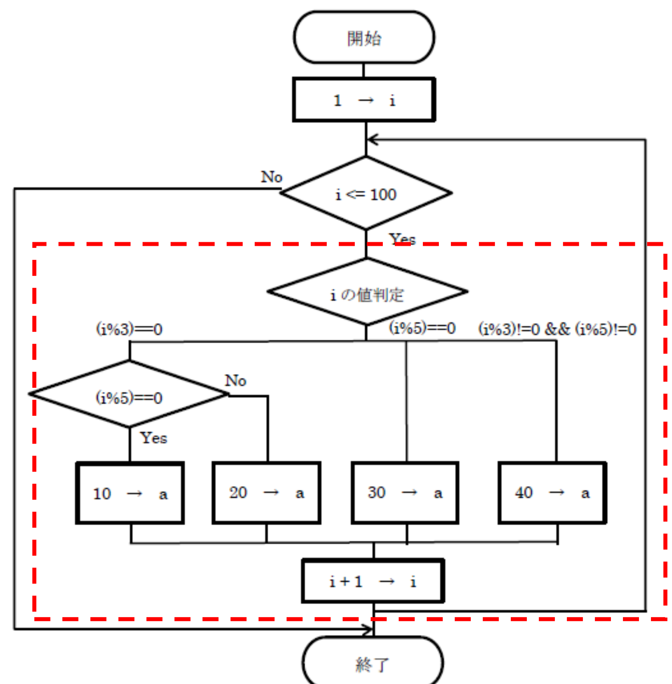

図 1 オリジナルのフローチャート

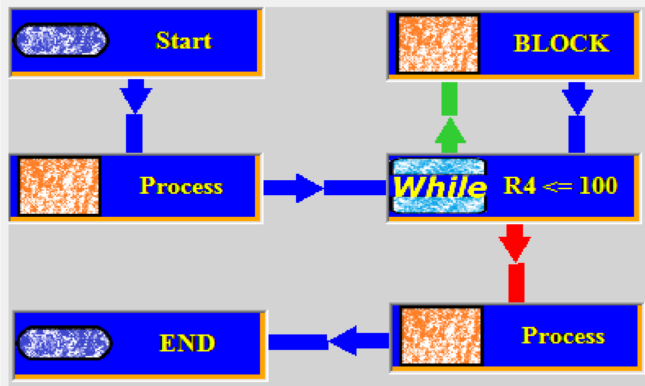

図 2 本解析ツールを用いて出力したフローチャート

図 2 は図 1 のフローチャートの全体を示しており，比較してみると，条件判定の部分が図 1 では “i<=100”，図 2 では “R4<=100”と表示されているが，フローチャートの各要素は対応付けが可能である．また，While-Loop の内部が “BLOCK”という名前で 1 つにまとまっている．これは，階層構造になっていることを示しており，これを展開すると内部構造を確認することができる．IF 文や While-Loop 文はその処理が階層構造になることがあるため，フローチャートのほうも，階層的に展開することができる仕様となっている．図 3 に図 2 の階層構造になっている”BLOCK”を展開したものを示す．

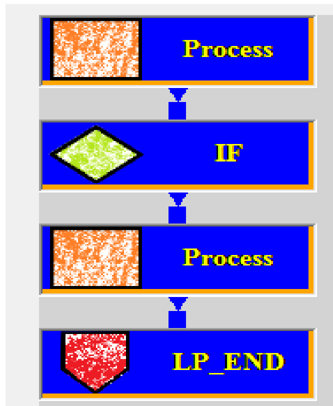

図 3 “BLOCK”を展開したフローチャート

図 3 は図 1 の点線で囲まれた部分に対応している．このように, IF 文と While-Loop 文は階層構造を用いて表現しているため，階層が深くなっても同様の方法でブロックを展開することで内部の構成を確認することができる．

3.1.2 動的解析部(プロファイリング)機能仕様

動的解析部では，静的解析部で作成したフローチャートを基に，命令を独自に開発した命令セットシミュレータ(ISS)で実行を行い，解析結果を出力するものである．詳細を以下に示す．また，本 ISS は ARM 社のホームページ掲載の ARM アーキテクチャリファレンスマニュアルなど[4] [5] [6]を基に作成したものである．

a. 入力仕様

入力される値は 32 ビットの ARM 命令のオブジェクトコードであり，1 命令ずつ ISS に命令コードを読み込ませ，解析を行う．

b. 解析仕様

本動的解析部では，ARM 命令の ISS を独自開発し，シミュレーションを行い，実行終了後に解析ビューとして解析結果を一覧表示するという仕様になっている．ユーザが指定したアドレスから解析を開始し，ユーザが指定したアドレス，または，最終命令までシミュレーションを行う．解析に用いる ISS は ARM 命令の主要 86 命令をサポートしている．ISS を用いて命令のシミュレーションを行うと同時に，各基本ブロックのコール回数，1 ブロック辺りのサイクル数，1 ブロック辺りの総サイクル数，全体の総サイクル数などを更新，インクリメントすることにより，各基本ブロックのプロファイリングを行う仕様となっている．

c. 出力仕様

出力方法は解析終了後に，解析ビューとして各基本ブロックのプロファイリング結果を出力する．出力内容について以下に一覧を示す．

- 基本ブロック ID
- 基本ブロック Type(Process, Decision, etc.)
- コード開始行
- コード終了行
- 行数
- 基本ブロックコール回数
- 基本ブロックの総サイクル数
- 基本ブロックの総消費電力
- 全体の総サイクル数

3.2 組込みコード解析ツールの GUI 出力仕様

本組込みコード解析ツールは C#言語で作成しており，操作はすべて GUI 上で行う．本ツールの GUI 出力表示を図 4 に示す．

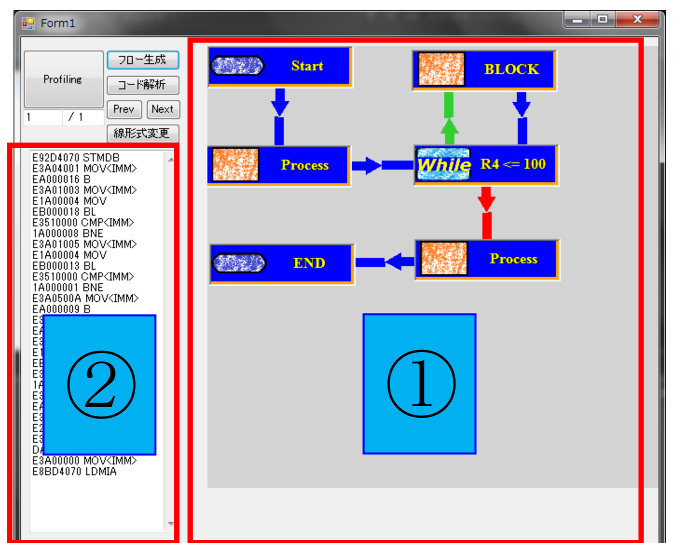

図 4 組込みコード解析ツールの GUI

図 4 は本組込みコード解析ツールの GUI 出力である．図中の①と②のエリアについて詳細を以下に示す．また，ボタン部分の詳細を図 4 に示す．

① フロー出力エリア

静的解析によって構成されたフローチャートを出力するエリア．エリア上で生成したフローをドラッグして移動することができ，見やすい位置に移動させることができる．

② マシンコード・アセンブリコード出力エリア

入力された命令マシンコードと解析の結果得られたアセンブリニーモニックを出力するエリア．マシンコードは 32 ビットで 16 進数表記で出力される．

図 5　組込みコード解析ツールボタン一覧

図 5 は本組込みコード解析ツールの処理ボタン一覧である．ボタンの説明を以下に示す．

① フロー生成ボタン

コード解析ボタンで入力，解析されたコードを元にフローチャートの生成を行う．フローチャートが 1 ページに収まらない場合は次のページにフローチャートを作成する．

② コード解析ボタン

オブジェクトコードをファイル入力し，各命令のマシンコードから命令を解析し，命令ニーモニック，レジスタ情報，IMM の値などを分析しテーブルに格納する．また，各命令が IF の何階層目であるか，Loop の何階層目であるかを付加する．

③ Prev ボタン

フローチャートのページを 1 つ前のページにする．1 ページ目で行うと，最終ページが表示される．

④ Next ボタン

フローチャートのページを 1 つ次のページにする．最終ページで行うと，1 ページ目が表示される．

⑤ 線形式変更ボタン

フローチャートを接続する罫線の形式を直線形式，折れ線形式に切り替える．

⑥ Profiling ボタン

動的解析を行う．プログラムの総ステップ数と実行時間を表示し，解析結果をファイルとして保存する．

4. ソフトウェアオブジェクトのプロファイリング手順

4.1 静的解析部

静的解析部ではフローチャートを生成するために 3 つのステップでフローチャートを生成している．その各ステップを以下に示す．

ステップ 1：命令解析

ARM7TDMI のオブジェクトコードの解析を行い必要な情報をテーブルに格納する．以下に解析情報を示す．

① 命令の種類

命令の種類とはオペレーションの種類であり，ADD，SUB などがその例として挙げられる．

② 使用レジスタ

命令によって使用されるレジスタの ID.

③ 使用 IMM データ

命令によって使用される IMM データ．

④ コンディション

命令を実行するための条件．コンディションが設定されている場合は命令ニーモニックの末端に，コンディション命令を追加する．

⑤ セットフラグ

コンディションフラグを更新することを示すフラグ．セットフラグの値が真である場合，さらに命令ニーモニックの末端に’S’を追加する．

ステップ 2：基本ブロック分割

命令の解析後，命令列を基本ブロックに分割する．基本ブロックは先頭から分岐命令，または，分岐命令の次の命令から，分岐命令までの命令列を 1 つの基本ブロックとし，命令列の分割を行う．

ステップ 3：フローチャート生成

分割された基本ブロックからフローチャートを生成する．出力内容は出力仕様に記述した通りである．また，フローチャート間の接続には 3 色の矢印を用いている．青色の矢印は接続先が 1 つしか無い場合の処理フローを示し，緑色の矢印は接続先が 2 つある場合かつ，条件判定が真であった場合の処理フローを示し，赤色の矢印は接続先が 2 つある場合かつ，条件判定が偽であった場合の処理フローを示している．

4.2 動的解析部

本組込みコード解析ツールは，静的解析でフローチャートを生成した後に，独自に開発した ISS を用いて動的解析を行う．解析のアルゴリズムを以下に示す．

ステップ 1：実行開始・終了アドレスの指定

ISS で命令を実行させるに先立って，開始アドレスと終了アドレスを指定する．

ステップ 2：コール回数をインクリメント

現在アクセスしている基本ブロックのコール回数をイ

ⓒ2016 Information Processing Society of Japan

ンクリメント．

ステップ 3：ステップ数のインクリメント

総サイクル数，1 ブロック辺りのサイクル数，1 ブロック辺りの総サイクル数をインクリメント．

ステップ 4：最大・最小サイクル数の更新

命令コードが基本ブロックの終端命令で，サイクル数が最大となった場合，1 ブロック辺りの最大サイクル数を更新し，最小であれば，最小サイクル数を更新する．

ステップ 5：命令コードの設定・実行

PC が指定したアドレスから命令コードを取得し，ISS で取得した命令コードを実行する．

ステップ 6：基本ブロックの終端命令まで実行

基本ブロックの終端命令までステップ 3 からステップ 5 を繰り返す．

ステップ 7：新たな基本ブロックを解析

新たな基本ブロックにアクセスし，ステップ 2 からステップ 6 までの処理を繰り返す．PC が終了アドレス，もしくは最終アドレスになった場合に処理を終了し解析結果を表示する．

5. 評価

5.1 解析評価結果

本組込みコード解析手法を用いて一般的なプログラムの解析を行った．解析を行った C プログラムの一覧を以下に示す．

- データ 1：クイックソート（QSort）
- データ 2：バブルソート（BSort）
- データ 3：平均値算出プログラム（Average）
- データ 4：ハノイの塔（Hanoi）
- データ 5：行列計算プログラム（Matrix）

上記で示したプログラムで解析を行う．まずはデータ 1 からデータ 5 に示すプログラムの詳細を表 1 に示す．

表 1　サンプルコードの詳細

	QSort	BSort	Average	Hanoi	Matrix
SubRoutine	2	1	1	2	1
BasicBlock	21	10	11	7	10
Instruction	83	47	75	28	78
Order	nlogn	n^2	n	2^n	n^3

表 1 中の項目の詳細を以下に示す．

SubRoutine：サブルーチン数(main を含む)

BasicBlock：基本ブロックの数

Instruction：命令数

Order：計算量

上記のプログラムを解析に用いる．本解析手法では，すべての基本ブロックにおいてプロファイリング結果を出力する．各プログラムのもっともサイクル数を要する基本ブロックについて詳細な解析結果を表 2 に示す．

表 2　プロファイリング結果

	QSort	BSort	Average	Hanoi	Matrix
ID	3	9	4	2	8
Start_Line	6	32	27	1	50
END_Line	16	37	29	4	67
Lines	11	6	3	5	18
Call	9	25	10	63	27
1B_TotalCycle	99	135	30	315	486
1B_TotalPower [nW]	13.86	18.9	4.2	44.1	68.04
TotalCycle	622	656	104	886	714
Rate [%]	15.92	22.87	28.85	35.55	68.07

表 2 中の項目の詳細を以下に示す．

ID：基本ブロック ID

Start_Line：コード開始行

END_Line：コード終了行

Lines：行数

Call：基本ブロックコール回数

1B_TotalCycle：基本ブロックの総サイクル数

1B_TotalPower：基本ブロックの総消費電力

TotalCycle：全体の総サイクル数

Rate：全体サイクル数内の占有率

本解析手法では，基本ブロックごとの全体サイクル数内の占有率が確認できるため，ボトルネックとなっている基本ブロックを検出できる．なお，消費電力は対象プロセッサの 1 サイクル辺りの消費電力を積算し求める．評価に用いた対象プロセッサの詳細を以下に示す．

対象プロセッサ：ARM968E-S

プロセス：TSMC130nmG

最適化タイプ：速度

消費電力：0.14[nW/Hz]

全体のサイクル数と比較して，各基本ブロック辺りの総サイクル数の占有率を示す帯グラフを図 6 に示す．

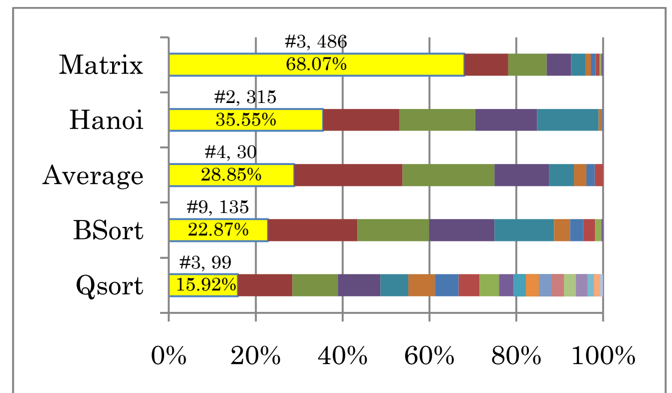

図 6　基本ブロックの占有率

図 6 は，各プログラムにおいて，そのプログラムを構成している各基本ブロックの占有率を表している．各プ

ログラムにおいて占有率の高い基本ブロックを左から順に並べ，黄，赤，緑，紫の順に色付けを行っている．図6上のもっともサイクル数を要する青のラベルに基本ブロックのID (#)，サイクル数，占有率を示す．

図6からクイックソートでは基本ブロックIDの3番がボトルネックであり全体の約16%を占めていることが確認できる．また，平均値算出プログラムでは基本ブロックIDの4番が全体の約29%を，行列計算プログラムでは基本ブロックIDの8番が全体の約68%を占めてボトルネックとなっていることが確認できた．行列計算プログラム($O(n^3)$)や，ハノイの塔($O(2^n)$)などの計算量が大きいプログラムに関しては，そこが大きく割合を占め，ボトルネックとなっているという傾向が確認できた．

5.2 性能評価結果

上記で示した5つのプログラムを100000回繰り返し，本動的解析手法のISS部分の性能評価を行った．実行環境を以下に示す．

CPU：Core i5-4440 CPU @3.10GHz
OS：Windows 7 Professional
対象プロセッサ: ARM7TDMI
実行環境：Visual studio 2013

表3に上記で説明した各プログラムの実行サイクル数，実行時間，解析速度を示す．

表3　各プログラムの処理速度

	実行サイクル数 [Mcycle]	実行時間 [sec]	解析速度 [MIPS]
QSort	62.2	37.62	1.65
BSort	65.6	40.30	1.63
Avarage	10.4	6.87	1.51
Hanoi	88.6	60.88	1.46
Matrix	71.4	49.86	1.43

本解析手法では，前処理としてプログラムを基本ブロック単位で分割し，分割したものを独自に開発したISSでシミュレーションを行い，命令を実行する度にサイクル数などの解析項目を積算する方法でプロファイリング結果を求めた．この前処理を除いたISSのシミュレーション時間と解析項目を積算する時間を実行時間とし，1秒辺りの解析命令数を解析速度として表3に示す．

表3から，すべてのプログラムにおいて1.4MIPS～1.7MIPS程度の実行速度が得られ一般的なISSとほぼ同等の結果をえることができた．

6. 結論

今回，オブジェクトコードからプログラムの構造を解析し，プロファイリングを行う組込みコード解析手法を提案した．組込みコード解析手法は静的解析部分と，動的解析部分に分けて解析を行っている．

静的解析部では，オブジェクトコードを基本ブロックに分割し，基本ブロックをフローチャートにすることでプログラムの構造が理解できるものを作成した．

動的解析部では, 各基本ブロックにおけるコール回数,実行サイクル数，総サイクル数を解析し，全体の総サイクル数と比較して，1 基本ブロックが全体サイクル数の何%を占めているのかをプロファイリングすることで，プログラムのボトルネック部分を検出することができ，プログラムの最適化を行ううえで有効であることを示すことができた．

本組込みコード解析ツールを用いて, ARMのオブジェクトコードをプロファイリングした結果，静的解析部分ではオブジェクトコードを基本ブロックに分割し，フローチャートを出力することで，オブジェクトコードの構造を確認することができた. また, 動的解析の部分では,基本ブロックごとにコール回数，サイクル数，総サイクル数をプロファイリングすることで，1つの基本ブロックの総サイクル数が全体サイクル数から占める割合を検出し，プログラムのボトルネック部分を検出できた，これにより，プログラムの最適化をサポートする手法の効果を確認することができた．

7. 今後の課題

今回は小規模なプログラムでの実験であったが，今後は多数の大規模なソフトウェアに適用してプロファイリングの効果を実証していきたい．また， 幅広くのオブジェクトコードを対象にプロファイリング，フローチャートの生成を行えるように，より汎用性の高いツールにして行くことも今後の課題である．Windows上で対象プロセッサのエミュレーションを行うことのできるQemu[7]のソースコードが一般公開されているため，これらの技術を応用することを考えている．

参考文献

1) 荒木大, “ELEGANTのSpecCシミュレーションと設計詳細化について”，第3回先端宇宙情報技術ワークショップ前刷公園集, pp.1-13, (2007-10)
2) Michiaki Muraoka, Noriyoshi Itoh, Rafael K. Morizawa, Hiroyuki Yamashita, Takao Shinsha, “Software Execution Time Back-annotation Method for High Speed Hardware-Software Co-simulation”, Proc. of SASIMI 2004, pp.169-175, October 2004
3) 青木一史, “リバースエンジニアリング入門(3)シェルコード解析に必携の「5つの道具」”， (2011-8) http://www.atmarkit.co.jp/ait/articles/1108/22/news110.html
4) “ARMアーキテクチャリファレンスマニュアル ARMv7-AおよびARMv7－Rエディション” (2009-3), ARM Limited
5) “ARM7TDMI(Rev4)テクニカルリファレンスマニュアル”, (2001-4), ARM Limited
6) “RealView Compilation Tools バージョン4.0 アセンブラガイド”, (2008-9), ARM Limited
7) “Qemu open souce processor emulator” http://wiki.qemu.org/Main_Page