SYSTEMORPH: Functionality Morphing

林田, 隆則
九州大学大学院システム情報科学府

數, 勇介
九州大学大学院システム情報科学府

村上, 和彰
九州大学大学院システム情報科学研究院

出版情報：SLRC 論文データベース, 2002-07. 組込みシステム技術に関するサマーワークショップ
バージョン：
権利関係：

SysteMorph : Functionality Morphing

林田 隆則[†], 數 勇介[†], 村上 和彰[‡]

arch@c.csce.kyushu-u.ac.jp [†, ‡]

あらまし 近年, システムを運用中に最適化する, 動的最適化技術に注目が集まっている. 我々は, SysteMorph という適応型動的最適化技術の概念を提案し, その応用システムの研究を行っている. 本論文では, SysteMorph とその応用のひとつである Functionality Morphing について述べる. また, 基盤技術のひとつであるオンライン・プロファイリング技術について, 再構成可能機能ユニットを搭載したプロセッサ・ベース・システムへの適用を考慮した HIS(Hot Instruction Sequence)プロファイラについて性能評価を行った. その結果, 単純な構造のプロファイラでも高い正確さで HIS 候補集合を収集できることが確認された.

1. はじめに

システムの最適化とは, システムの性能や消費電力などについて, 要求を満たすようにその特性を変更することである. これまで, 設計時におけるシステムの最適化についてさまざまな研究がなされてきた. さらに, 近年の半導体技術の進歩による回路規模の増大やシステムの複雑化により, システムの設計時には想定できない状況に対応する最適化が重要視されるようになってきた. そこで, 近年システムの運用中に動作状況に応じてそのシステムを最適化する, 動的最適化技術が注目されている.

動的最適化には, システムを運用することで初めて得られる情報を用いてシステムを最適化できるという大きな利点がある. これにより, 入力によってシステムの振舞いが異なるようなシステムに対して, 入力に応じて最適なシステムの形態をとるようなシステムの実現が可能となる.

しかしながら, 動的最適化には克服すべき課題も多い. システムの運用中に最適化を施すには, できる限りそのシステムの本来の動作に影響を与えないように最適化を実現しなければならない. また, システムの動作中に最適化すべきポイントを特定する必要もある. さらに, 特定した最適化ポイントに対して, 実際に最適化をどのように行うかも考えなければならない. また, システムを変更できるように, システムを再構成するための機構もあらかじめ組み込まれている必要がある.

このような動的最適化可能なシステムについて, 我々は SysteMorph という概念を提案する.

SysteMorph とは, 主にプロセッサ・ベース・システムにおいて, 対象とするアプリケーションのプロファイル情報に基づいて, システムを適応的かつ動的に最適化する技術である. SysteMorph の基盤技術は, (1)オンライン・プロファイリング技術, (2)適応型動的最適化技術, (3)スマート・ハードウェア技術の 3 つに大別できる.

本論文では, SysteMorph の応用のうち, 再構成可能ハードウェアを用いた最適化を行うFunctionality Morphingを取り上げ, Functionality Morphing への適用を考慮したオンライン・プロファイリング技術に注目する. 再構成可能な機能ユニットを持つプロセッサを想定し, その機能ユニットをアプリケーションについて最適に活用できるような情報をアプリケーションの実行時に収集するオンライン・プロファイリング技術について検討を行う.

2 章で我々が提案するSysteMorph について述べ, 3 章で SysteMorph の応用のひとつである Functionality Morphing について述べる. 4 章でオンライン・プロファイリング技術について, その概論とFunctionality Morphing への適用を考慮したオンライン・プロファイリングについて述べ, 5 章でオンライン・プロファイラの性能について実験と

[†]九州大学大学院システム情報科学府
[‡]九州大学大学院システム情報科学研究院
[†‡]〒816-8580 福岡県春日市春日公園 6-1

評価を行う. 6 章で関連研究を紹介し, 7 章でまとめる.

2. SysteMorph

2.1. SysteMorph とは

SysteMorph とは,「対象とするアプリケーション・プログラムの実行時プロファイル情報に基づいて, 当該アプリケーションの実行中に

・ソフトウェア

・命令セット・アーキテクチャ

・ハードウェアのモードならびに構成

を変更して, その性能, 消費電力, 消費エネルギーを最適化する」, すなわち Feedback-Directed Dynamic Software / Instruction Set Architecture / Hardware Co-optimization 技術である. (図 1)

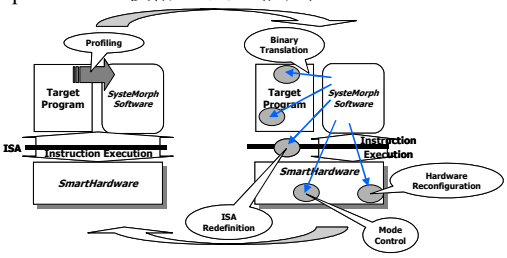

図 1 SysteMorph の概念

SysteMorph は, 以下の 3 つの基盤技術から成る.

・オンライン・プロファイリング技術

・適応型動的最適化技術

・スマート・ハードウェア技術

オンライン・プロファイリング技術とは, アプリケーションの実行時にプロファイル情報を取得し, さらに得られた情報から最適化に必要な情報を抽出して最適化機構へ適宜フィードバックする技術である. 適応型動的最適化技術とは, ソフトウェアのコード変換やハードウェアの再構成などの技術を用いて, プロファイリングによって得られた情報を基にシステム全体を動的に最適化する技術である.

スマート・ハードウェア技術とは, ハードウェアに対して適応型動的最適化技術を適用するために, ハードウェアの再構成やモード変更を可能にするためのハードウェア構成技術である.

2.2. 実装の選択肢

SysteMorph では, 3 つの基盤技術である, オンライン・プロファイリング技術, 適応型動的最適化技術, スマート・ハードウェア技術のそれぞれについて, 特定の技術を想定しない. そのため, 各技術の実装に自由度がある.

例えば, 組み込みシステムにおいて SysteMorph の応用を考えた場合, システム内部に動的最適化のための機構を全て取り込むのはシステムの肥大化につながるため好ましくない. したがって, システム内部には実行時にプロファイルを取得する機構, および, 再構成可能な機構のみを実装し, 最適化のプロセスは外部のシステムに委託するというシステムが考えられる.

携帯電話を例に SysteMorph の応用を考えると, 次のようなシステム構成が考えられる.

システムは, 機器側(この例では携帯電話端末)と, 外部システム(この例では最適化を行うセンター)から構成される. 機器側はまず動作中のアプリケーションのプロファイルを収集する. 電力を十分に使える充電時に, 外部システムに収集したプロファイルを送信する. 外部システムは得られたプロファイルから最適化を行う. 機器側は, 次に外部システムと通信する際に, 自身の再構成に関する情報があるかどうかも問い合わせ, それが存在すれば, その情報を基に自身を再構成する.

上記は一例であり, SysteMorph の応用の自由度は,

- 実行時再構成／アイドル時再構成
- Repeatedly Morphing／One-time Morphing
- Parallel Morphing／Concurrent Morphing
- In-Suite Optimization／Out-Suite Optimization

などが挙げられる. これらの中から, システム設計時に特性に合わせた応用を選択することができる.

先の例は, この区分上では Repeatedly Concurrent Morphing by Out-Suite Optimization となる.

3. Functionality Morphing

Functionality Morphing とは, SysteMorph の応用のうち, プロセッサと再構成可能ハードウェアを搭載したシステムにおいて, システムの稼動中にハードウェアの再構成を行い, かつ, その再構成したハードウェアを活用するようにソフトウェアを変更することによってシステムを最適化するものである. これを実現するためには, システム上で稼動しているアプリケーションに対する実行時プロファイルの取得

と最適化部分の特定，ハードウェアの実行時再構成，などの課題を解決する必要がある．

Functionality Morphing を応用したシステムの選択肢としては，

- 再構成可能ハードウェアの配置箇所
- 再構成可能ハードウェアの構成
- 再構成のタイミング
- プロファイラの構成
- プロファイルの粒度

などが挙げられる．

再構成可能ハードウェアの配置箇所としては，プロセッサ内部の演算器と同様にレジスタへ直接アクセスできる位置や，コプロセッサのようなプロセッサの外部の位置が考えられる．

また，再構成のタイミングとしては，アプリケーションの実行中に並行して行う方法と，アプリケーションの実行が停止しているときに行う方法が考えられる．

4. オンライン・プロファイリング

オンライン・プロファイリングは，アプリケーションの振舞いに関する情報をその実行中に収集する技術である．

オンライン・プロファイリングに対して，アプリケーションの全体の振舞いに関する情報を収集する手法をオフライン・プロファイリングと呼ぶ．一般的に，単にプロファイリングといった場合はオフライン・プロファイリングをさす場合が多いが，本稿では，とくに断らない限り「プロファイリング」はオンライン・プロファイリングを指し，「プロファイル」はオンライン・プロファイリングにより収集された情報のことを指すこととする．

オンライン・プロファイリングに対する最大の課題は，いかにアプリケーション本体の実行に影響を与えずに，かつ正確なプロファイルを取得するかである．

オンライン・プロファイリングの利点としては，現在実行しているアプリケーションに対するその時点での入力列に対応したプロファイルを取得することができることが挙げられる．しかし，実行時最適化を目的にオンライン・プロファイリングを行う場合，アプリケーションの実行中に，その時点までに得られたプロファイルからその後のアプリケーションの動作を予測し，その予測に基づいて最適化のヒントとなる情報を検出する必要がある．

一般に，予測に使用する情報が多ければ多いほど予測の精度は高くなる．オンライン・プロファイリングにおいても，アプリケーションの開始から時間が経過するほどプロファイル情報が多くなり，その後のアプリケーションの動作に関する予測の精度が高くなると考えられる．

しかしながら，予測の精度を上げるために長時間のプロファイリングを行うと，最適化後の実行時間が少なくなり，最適化の恩恵を受ける割合が減少する．そのため，オンライン・プロファイリングでは「高速」かつ「正確」にアプリケーションの動作を予測するための情報を取得する必要がある．

4.1. オンライン・プロファイリングの分類

一般的に，オンライン・プロファイリングは，プロファイリングの対象となるイベントの列を収集する．オンライン・プロファイラは，プロファイリングを行う機構や，プロファイリングのポリシーによって以下のように分類できる．

- Profiling by Software / by Hardware / by Hybrid
- Precise / Imprecise Profiling
- Event Stream Compression / Non-Compression
- Lossy Compression / Lossless Compression

Precise Profiling は，注目しているイベント列を完全に監視し，プロファイルを収集するプロファイリング手法である．一方，Imprecise Profiling は注目するイベント列の一部を（すなわち，不完全に）監視し，プロファイルを収集する手法である．

Event Stream Compression は，収集したイベント列に関する情報を圧縮して記憶する（もしくはメッセージとして外部へ通知する）手法である．この手法は，圧縮の仕方でさらに分類できる．Lossy Compression では，圧縮時の情報欠落を許す．したがって，Lossy Compression を用いた手法では，元のイベント列を復元することはできない．一方，Lossless Compression を用いる場合は，Precise Profiling と組み合わせた場合，もとのイベント列を完全に復元することが可能になる．

4.2. オンライン・プロファイリングの対象

オンライン・プロファイリングの対象としては，代表的なも

のに次のようなイベントの列が挙げられる.

- 頻繁に実行される命令が格納されているアドレスへのアクセス
- 頻繁に実行される命令, もしくは命令列の実行
- 頻繁に実行されるアプリケーション中のパスの実行

アドレスを基本とするプロファイリングの場合, 内容が同一でも格納箇所が異なればそれらは別のものとして扱われる. アプリケーション内で頻繁に発生する同一, もしくは類似した処理を抽出する場合, アドレスを基本とするプロファイリングよりは内容を考慮するプロファイリングの方が向いているといえるが, 後者のプロファイラは複雑さが増すという短所もある.

4.3. オンライン・プロファイラへの要求

オンライン・プロファイラはその性質上, 次のような要求を満たさなければならない.

- オーバヘッドを削減する
- コストを削減する
- 振舞いを正確に予測する

オンライン・プロファイラはアプリケーションの実行時にアプリケーション本来の動作と並行に動作する. しかし, オンライン・プロファイラはアプリケーションの振舞いを監視するものであり, アプリケーションの実行そのものに直接影響を与えるものではない. したがって, オンライン・プロファイラを導入したことによるアプリケーション本来の動作の性能低下をできるだけ少なくする必要がある. また, オンライン・プロファイラはシステムに組み込まれるため, オンライン・プロファイラにかかるソフトウェア的なコスト(プログラムサイズの増大など), ハードウェア的なコスト(プロファイラのための回路部分の面積など), 消費電力面のコスト(ソフトウェアやハードウェアの追加による消費電力の増大)などをできるだけ少なくすることが要求される.

さらに, アプリケーションの実行中にハードウェアやソフトウェアの再構成を行うためにプロファイルからそのヒントを抽出する場合, オンライン・プロファイラは実行しているアプリケーションのその時刻以降の振舞いを予測して最適化のヒントを見つけなければならない. その際, 予測が正確でなければ再構成後のシステムはそのアプリケーションの実行に対して最適なものにはならない. したがって, 再構成後のシステムをアプリケーションの実行について最適なものへ近づけるためには, 予測の正確さを向上させることが重要である.

4.4. Hot Instruction Sequence プロファイラ

本論文では, オンライン・プロファイラの中で, 頻繁に実行される命令列をプロファイリングする Hot Instruction Sequence (HIS) プロファイラについて議論する. 本論文で議論するプロファイラは, 前述の分類において, Hardware Precise Profiler with Lossy Event Stream Compression に相当する.

HIS とは, アプリケーションにおいて頻繁に実行される命令の列のことをいう. 命令列が格納されているアドレスが異なっても, 出現する命令の列が同一であればそれらは同一のものとみなされる.

HIS プロファイラの構成としては, 以下のようなものが考えられる.

- サンプリング型 HIS プロファイラ
  - 周期サンプリング・HIS プロファイラ
  - ランダム・サンプリング・HIS プロファイラ
- イベント・カウント型 HIS プロファイラ

周期サンプリング・HIS プロファイラは, プロセッサが直前に実行した L 命令(L はプロファイリングする命令列の長さ)を記憶しておき, 周期 R でその L 命令を最適化機構へメッセージとして送信する.

ランダム・サンプリング・HIS プロファイラは, 周期サンプリング・HIS プロファイラの周期がランダムになったものである. 周期 R の周期サンプリング・HIS プロファイラと同等のサンプリングを行うためには, 確率 1/R でサンプリングを行えばよい. なお, ハードウェアでの実装を考える場合には, 擬似ランダム値として周期カウンタを 1 から2 R までのランダムな値に設定した後, 命令が実行されるたびに周期カウンタをデクリメントし, 周期カウンタが 0 になったときにサンプリングを行うようにすれば, 周期 R の周期サンプリングに相当する擬似ランダム・サンプリングを実現することが可能である.

イベント・カウント型 HIS プロファイラはカウントする命令列とカウント値の組を記憶するテーブルを用いて, プロセッサが実行した各命令列の頻度をカウントする. ある命令列

が実行されたとき，プロファイラはその命令列の実行をすでにカウント開始しているかどうかをチェックする．もしすでにカウントを始めている場合，そのカウント値をインクリメントする．そうでない場合，テーブルに空きエントリが存在すればそこにその命令列を登録し，空きエントリがなければ何らかのアルゴリズムにしたがってエントリの追い出しを行う．本論文で議論するプロファイラでは，エントリの追い出しの際にそのエントリの情報を外部に出力しない．

5. 実験

4.4 節で述べた HIS プロファイラについて，それぞれ評価モデルを作成し，プロファイリングの正確さについて測定を行った．

5.1. 実験環境

実験には，SimpleScalar ToolSet Ver.2.0[5]を用いた．また，プロファイリングの対象プログラムとして SPECCPU 95[6]の整数ベンチマークの中からいくつかのベンチマーク・プログラムを選択して使用した．なお，各ベンチマーク・プログラムは SimpleScalar ToolSet 付属の gcc で-O2 オプションをつけてコンパイルした．

また，全ての実験はトレースベースで行った．SimpleScalar でベンチマーク・プログラムのトレースを作成し，そのトレースから全実行命令数と次節で述べる真の HIS 集合を求める．その後，各評価モデルにおいて，各時刻の評価指標値 Hitrate，および Noiserate（詳細は次節）を計算する．

5.2. 評価モデル

本実験では，

- 周期サンプリング・HIS プロファイラ・モデル
- ランダム・サンプリング・HIS プロファイラ・モデル
- イベント・カウント型 HIS プロファイラ・モデル

を定義する．

全てのプロファイラ・モデルは，命令履歴バッファという直前に実行した L 命令を記憶しておくバッファを持つ．このバッファは，1 命令実行ごとに更新される．

周期サンプリング・HIS プロファイラ・モデルは，R 命令実行ごとに命令履歴バッファに含まれる命令列を HIS と仮定して，HIS 候補集合に追加する．R はパラメータとして可変とする．

ランダム・サンプリング・HIS プロファイラ・モデルは，1 命令実行ごとに確率 1/R で命令履歴バッファに含まれる命令列を HIS と仮定して HIS 候補集合に追加する．R はパラメータとして可変とする．

イベント・カウント型 HIS プロファイラ・モデルは，命令履歴バッファとカウント・テーブルを持つ．テーブルの各エントリの要素は命令列とカウント値である．命令履歴バッファが更新されると，その内容と同じ命令列を持つテーブルのエントリを検索する．検索でヒットすれば，そのエントリのカウント値をインクリメントする．カウント値が一定の閾値を超えると，その命令列を HIS と仮定し，HIS 候補集合に追加する．

各評価モデルに与えるパラメータを表 1 に示す．

表 1 評価モデルのパラメータ

評価モデル	パラメータ名	パラメータ値
全モデル共通	HIS 閾値	全実行命令数の 0.1%
	HIS 長	4命令
周期サンプリング型	サンプリング周期	32,64,128,256, 512,1024 命令実行
ランダム・サンプリング型	サンプリング確率	1/32,1/64,1/128, 1/256,1/512,1/1024
イベント・カウント型	テーブルサイズ	256,512,1024, 2048 エントリ
	HIS 候補確定閾値	256
	リプレース・ポリシ	LRU

なお，本実験において，命令列は OP コード部分のみに注目しており，オペランド部分の違いは無視している．したがって，OP コード部分が同一でオペランド部分のみが異なる命令列はすべて同一のものとして扱われる．これは，複雑さを避けるためと，オペランドの違いは演算機に与えるデータの違いであり，再構成時に吸収できると考えたためである．また，プロファイラはプロセッサの処理と完全に

独立して動作し，性能に対するオーバヘッドはないものと仮定する．

これらの評価モデルの正確さの評価指標として，HitrateとMissrateをそれぞれ定義する．Hitrateは，真のHIS集合に含まれる命令列に対して，その実行頻度で重み付けをした場合に，HIS 候補集合内の命令列がどれくらい真のHIS をカバーしているかを表す指標である． Missrate はHIS 候補集合の中に含まれていて，真の HIS に含まれない命令列の割合に関して，Hitrate 同様重み付けを考慮して表す指標である．詳細な定義は以下のとおりである．

s : 任意の命令列

S : 任意の命令列の集合

T_{fin} : アプリケーションの終了時刻

D(*t*)= {時刻*t*までにHIS候補列に追加された命令列}

F = {真のHIS集合に含まれる命令列}

freq(*s*,*t*) = {時刻 *t* までの命令列 *s* の実行回数}

$$freq(S,t) = \sum_{s \in S} freq(s,t)$$

$$HitRate(t) = \frac{\{freq((D(t) \cap F), Tfin) - freq((D(t) \cap F), t)\}}{freq(F, T_{fin})}$$

$$NoiseRate(t) = \frac{\{(freq(D(t) - F), T_{fin}) - freq((D(t) - F), t)\}}{freq(F, T_{fin})}$$

5.3. 実験結果および考察

図2から図7は，ベンチマーク・プログラム(132.ijpeg)に関して，各プロファイラの Hitrate と Noiserate を求めたものである．グラフは横軸が実行命令数，すなわち時刻の経過を表しており，縦軸はHitrateおよびNoiserateである．ただしNoiserateのグラフは上端が 0.01 または 0.02 である．

図 2，図4，および，図 6 からわかるように，Hitrate についてはどのモデルにおいてもほぼ同様の結果を示しており，アプリケーションの開始から全実行時間の 10%以内の点においてHitrateはピークを迎え，以降はHitrateが減少する．Hitrate は真の HIS 集合に含まれる命令列のその時刻以降の実行回数も影響するため，時間が経過するほどHitrateの上限値は減少する．

また，Hitrate についてはほとんどパラメータによる違いが出ていない．これは，アプリケーションの前半で以降に実行する命令列をHIS候補に含めるのに十分な回数実行しているからだと考えられる．

一方，図 3，図 5，および図 7 から，Noiserate については各モデルで傾向は似ているものの，とくにサンプリング型とイベント・カウント型で最悪のNoiserateの値に大きな差が生じている．これは，サンプリング型がある特定のタイミングで命令列が履歴バッファに乗った場合にその命令列をHIS 候補に含めるのに対して，イベント・カウント型はテーブルを用いて実行回数の少ない命令列を「足切り」するために，Noiserate に影響する命令列をHIS候補に含めないためであると考えられる．また，イベント・カウンティング型HIS プロファイラにおいて，テーブルサイズが大きいモデルの方が高いNoiserateを記録している．この原因としてはテーブルが必要以上に大きいために，真の HIS には含まれないが HIS 候補として確定する閾値よりは多く実行されるような命令列がエントリの追い出しを受けずにテーブルに残ってしまったためと考えられる．このことから，テーブルサイズについては大きすぎると逆にプロファイリングの精度を落とす結果になる．

グラフで示したベンチマークにおいては顕著に現れていないが，Noiserate の増大を防ぐために必要以上にテーブルを縮小すると，エントリの追い出しが頻発し Hitrate が下がってしまう可能性もあるため，テーブルのサイズについては一概に最適なサイズを決定することはできないことがわかる．これについては，システム設計時にオフライン・プロファイリングを活用してテーブルサイズを決定するなどの手法で対応することが考えられる．

また，テーブルのサイズが大きすぎることによるNoiserate の増大には，活性化するサイズを変更できるようなテーブルを用いたプロファイラによって Hitrate を高く保ちながら Noiserate を低減できると考えられる．この機構を持つプロファイラについては今後評価を行う予定である．

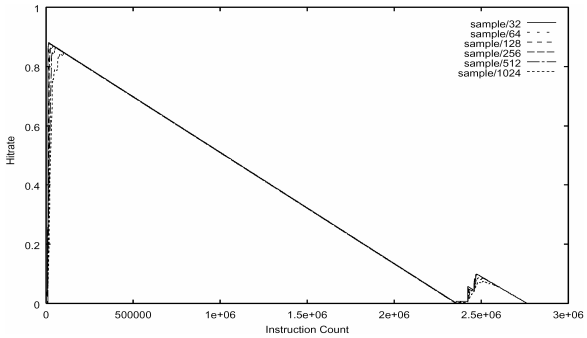

図 2 周期サンプリング型 HIS プロファイラの Hitrate

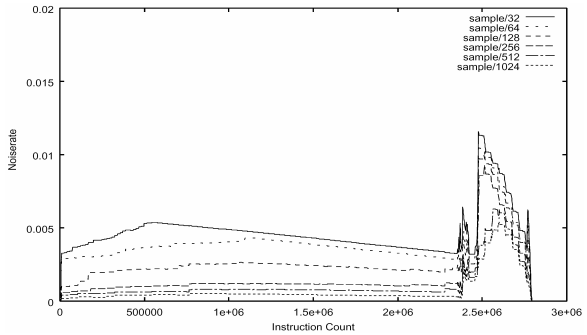

図 3 周期サンプリング型 HIS プロファイラの Noiserate

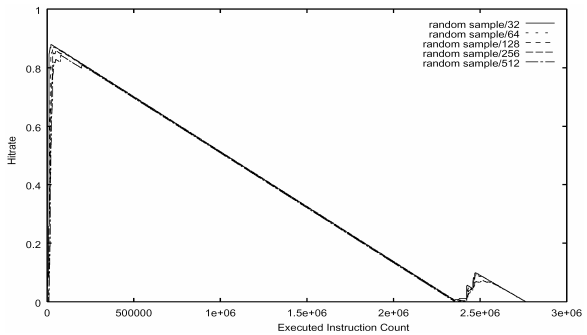

図 4 ランダム・サンプリング型 HIS プロファイラの Hitrate

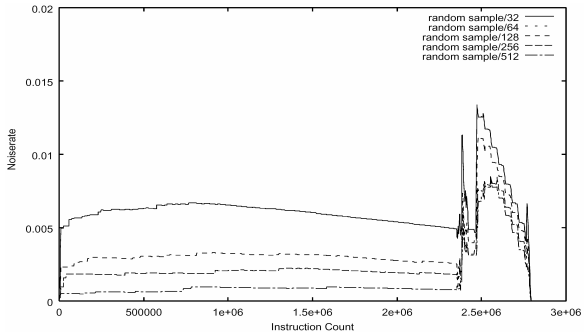

図 5 ランダム・サンプリング型 HIS プロファイラの Noiserate

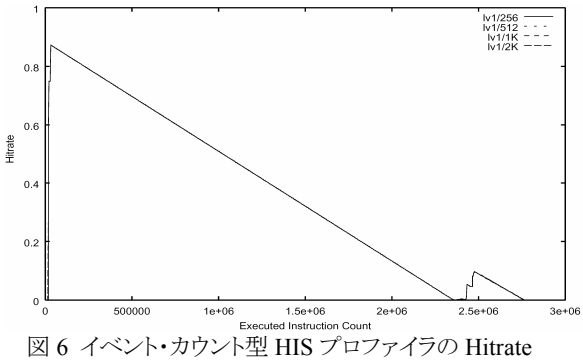

図 6 イベント・カウント型 HIS プロファイラの Hitrate

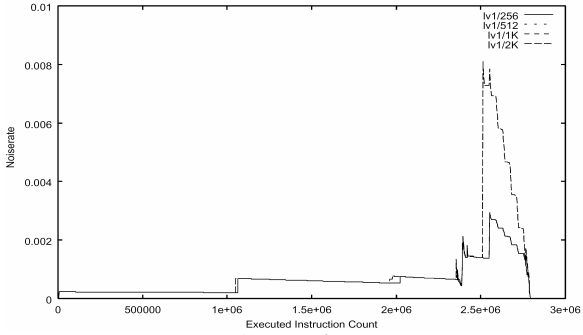

図 7 イベント・カウント型 HIS プロファイラの Noiserate

6. 関連研究

オンライン・プロファイリングに関して，近年さまざまな研究がなされている．

Sastry らは，論文[1]において階層構造のサンプリングにおいて高速なオンライン・プロファイリングを実現する手法を提案している．本論文におけるサンプリング型のプロファイラは，論文[1]における単純なサンプリング型のプロファイラに属する．また，論文[1]では複雑な階層構造をもつサンプリング型のプロファイラも提案されている．本論文では評価していないが，これらの形態を持つ HIS プロファイラについて，今後性能を評価する予定である．

Evelyn ら[2]は，アプリケーション中のパス，すなわち連続して実行される基本ブロックの集合について，頻繁に実行されるパスをアプリケーションの実行時に推定する手法として NET(Next Execution Tail)法を提案している．この手法は，内容が同一であってもアドレスが異なればそれらは異なるものとして処理される．本論文で提案している HIS はアドレスに依存しないので，その点が異なる．Functional Morphing の応用として再構成可能機能ユニットよりも大規

模な再構成可能部分 (再構成可能コプロセッサの利用など) を考慮した場合, この手法を Functional Morphing に応用したパス・プロファイリングの手法の適用が考えられる.

Craig ら[3]は, ソフトウェアでのプロファイリングによるオーバヘッドを低減するために, プロファイリングを専門に行うプログラマブル・コプロセッサについて検討を行っている. 論文中では, ロード命令によりロードされた値と, 値予測の結果をプロファイリングする手法を提案している. 専用のコプロセッサを用いることによりオーバヘッドが 30~45%削減できると述べており, ハードウェアによるプロファイリングのオーバヘッド削減効果は非常に大きいことがわかる.

Dhodapkar ら[5]は, アプリケーション中に高頻度でアクセスされる部分(Working Set)はアプリケーション内の「フェイズ」によって変化するという考えから, そのフェイズの変化を Working Set の変化からフェイズの変化を発見する手法について提案している. この手法を応用すると, 複数アプリケーションが実行されるような Functional Morphing 応用システムにおいて再構成部分の構成情報の切り替えなどに活用できると考えられる.

7. おわりに

本論文では, 適応型動的最適化システムの概念である SysteMorph を提案し, SysteMorph の応用のひとつとして Functional Morphing について述べた.

また, Functional Morphing の応用のひとつである再構成可能機能ユニットを用いたプロセッサ・ベース・システムへの適用を考えたオンライン・プロファイリング技術について, HIS プロファイラを例にいくつかの実現手法を述べ, それらの正確さについてベンチマーク・プログラムを用いて測定を行った. その結果, テーブルを用いた単純な構成のプロファイラで, アプリケーションの全実行時間の 10%以内に最大の Hitrate を与える HIS 候補の集合を得られることがわかった.

本論文では, 単一のアプリケーションにのみ着目して測定を行ったが, 今後, 複数のアプリケーションが並行に, あるいはシーケンシャルに動作する場合における, これらのプロファイラの有効性や, 命令列以外を対象にしたオンライン・プロファイリング技術などについて検討を行い, SysteMorph に活用可能なオンライン・プロファイリング技術について研究を進める. また, SysteMorph を構成するほかの基盤技術についても研究を進め, システムとして確立した SysteMorph の応用システムの研究を進める.

謝辞 本研究は一部, 日本学術振興会科学研究費補助金基盤研究(A)(2)展開研究「システム LSI 向けカスタム化可能 IP コアのアーキテクチャおよび設計支援技術の開発」(課題番号:12358002), 日本学術振興会科学研究費補助金基盤研究(A)(2)一般研究「ハードウェア構成を動的に最適化する「ハードウェア・モーフィング」技術の開発」(課題番号:13308015)による. 日頃からご討論頂く九州大学安浦・村上・松永研究室の諸氏に感謝致します.

参考文献

[1] "Rapid Profiling via Stratified Sampling", S. S. Sastry, R. Bodik and J. E. Smith, The 28th Annual International Symposium on Computer Architecture (ISCA2001), Jul., 2001.

[2] "Software Profiling for Hot Path Prediction: Less is More", E. Duesterwald and V. Bala, Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2001), pp202-211, Nov., 2000.

[3] "A Programmable Co-Processor for Profiling", C. B. Zilles and G. S. Sohi, Proceedings of Seventh International Symposium on High Performance Computer Architecture (HPCA 2001), pp241-252, Jan., 2001.

[4] "Managing Multi-Configuration Hardware via Dynamic Working Set Analysis", A. S. Dhodapkar and J. E. Smith, The 29th Annual International Symposium on Computer Architecture (ISCA 2002), May, 2002.

[5] SimpleScalar Toolset http://www.simplescalar.com/

[6] Standard Performance Evaluation Corporation http://www.spec.org/