Vol.2010-SLDM-146 No.3
2010/10/5

マイクロプロセッサにおける制御信号系列の誤りを検出する動的シグネチャ検査技術

杉原　　真[†,††]

† 豊橋技術科学大学大学院工学研究科情報・知能工学系，〒 441-8580 愛知県豊橋市天伯町雲雀ヶ丘 1-1
†† 独立行政法人科学技術振興機構, CREST, 〒 102-0075 東京都千代田区三番町 5 番地
E-mail: †sugihara@tut.jp

あらまし　集積回路の微細加工技術が進歩するにつれ，ソフトエラーや NBTI (negative bias temperature instability) といった信頼性に関わる問題が顕在化してきた．今後，民生品の設計においても信頼性要求を考慮することが益々重要となる．本稿では，高い信頼性を有するコンピュータシステムを実現するための動的シグネチャ検査技術 (DCSM: dynamic continuous signature monitoring) を提案する．DCSM 技術はプログラムの実行中に実行時シグネチャを生成すると共に参照シグネチャも生成する．既存の静的シグネチャ検査技術 (SCSM: static continuous signature monitoring) はプログラム/データメモリ中に参照シグネチャを保存しておくが，提案する DCSM 技術は，ハードウェア中に実装されているシグネチャテーブルにシグネチャを保存する．動的な参照シグネチャの生成により，プログラムメモリあるいはデータメモリに参照シグネチャを保存しておく必要がなく，使用メモリ量を削減できる．計算機実験により，シグネチャテーブルの大きさに応じて，実行される命令の 1.4-100.0%が DCSM 技術によって保護できることを確認した．

キーワード　ソフトエラー，SEU (single event upset)，SET (single event transient)，NBTI (negative bias temperature instability)，制御信号，信頼性，脆弱性，マイクロプロセッサ

A Dynamic Continuous Signature Monitoring Technique for Reliable Microprocessors

Makoto SUGIHARA[†,††]

† Department of Computer Science and Engineering, Toyohashi University of Technology,
1-1 Hibarigaoka, Tempakucho, Toyohashi, Aichi 441-8580 Japan
†† Japan Science and Technology Agency, CREST,
5 Sanbancho, Chiyoda-ku, Tokyo 102-0075 Japan
E-mail: †sugihara@tut.jp

Abstract　Reliability issues such as a soft error and NBTI (negative bias temperature instability) have become a matter of concern as integrated circuits continue to shrink. It is getting more and more important to take reliability requirements into account even for consumer products. This paper presents a dynamic continuous signature monitoring (DCSM) technique for high reliable computer systems. The DCSM technique dynamically generates reference signatures as well as runtime ones during executing a program. The DCSM technique stores the generated signatures in a signature table, which is a small storage circuit in a microprocessor, unlike the conventional static continuous signature monitoring (SCSM) techniques. The DCSM technique contributes to saving program or data memory space that stores the signatures. Our experiments showed that our DCSM technique protected 1.4-100.0% of executed instructions depending on the size of signature tables.

Key words　Soft Error, SEU (single event upset), SET (single event transient), NBTI (negative bias temperature instability), Control Signal, Reliability, Vulnerability, Microprocessor

1. はじめに

微細加工技術が進展するにつれ，ソフトエラーや NBTI (negative bias temparature instability) といった信頼性に影響を及ぼす現象が問題になりつつある．今後は宇宙向け IC (integrated circuit) のみならず，民生機器向け IC においても信頼性を考慮し設計を行うことが求められる．伝統的な高信頼化技術として，回路を多重化する手法，及び冗長な符号化による手法が挙げられる．これらの手法による面積ペナルティ及び性能ペナルティは大きく，民生機器向け VLSI に適用する上では課題となる．民生機器向け IC の高信頼化を達成する上では，低コスト性が求められる．

シグネチャ検査 (CSM: continuous signature monitoring) はプログラム実行時に制御信号誤りを検出する技術である [17]．CSM 技術は，決まって出現する信号値系列（例えば，基本ブロックにおける PC の系列）に対応付けられる値，すなわちシグネチャを用いることで実現される．CSM 技術の典型的な一例では，一定の信号値系列に対してプログラムのコンパイル時に**参照シグネチャ(reference signature)** を計算し，それらをプログラムコード中に埋め込む．プログラム中に埋め込まれた参照シグネチャとプログラム実行中に計算された**実行時シグネチャ (runtime signature)** を比較することで，制御フロー誤りの有無を判定する．図 1 に，単純な CSM の一例を示す．この例は，実行命令列に対して CSM を適用したものである．シグネチャコンパイラはアセンブリコードを基本ブロックに分割し，各基本ブロックの参照シグネチャを計算する．算出された参照シグネチャと，プログラム実行時に算出される実行時シグネチャを比較するために，シグネチャ命令が基本ブロックの末尾に追加される．シグネチャ命令はオペコード部とシグネチャ部からなる．プログラム実行時に，専用ハードウェアにより基本ブロックに対応する実行時シグネチャが生成される．各シグネチャ命令において，参照シグネチャと実行時シグネチャが比較され，制御フロー誤りがあれば検出される．

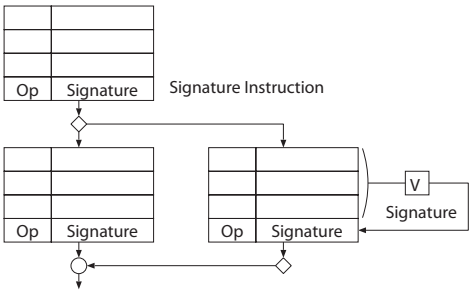

図 1　CSM 技術の一例.

CSM 技術に関する研究は 1980 年代より行われている [7], [9], [11], [17]．静的に，すなわちプログラムのコンパイル時などのプログラム実行開始以前に，参照シグネチャを求めプログラムメモリに埋め込む CSM 技術においては，メモリオーバヘッド，及び制御フロー誤りの検出率などが評価尺度となる．Namjoo の手法 [7]，Shen らの手法 [9]，及び Wilken らの手法 [17] はそれぞれ 12-21%，6-15%，及び 4-11%のメモリオーバヘッドを生じ，それらの制御フロー誤りの検出率はそれぞれ 99.5-99.9%，85-93%，及び 99.9999%である [17]．いずれの手法も高いメモリオーバヘッドを生じるものであり，メモリオーバヘッドを抑制することが望ましい．また，上記のいずれの手法もプログラムに対してなんらかの加工を行うために，CSM 技術が適用されているプログラムはハードウェアの互換性を失う．ハードウェアの互換性喪失は，プログラムの互換性が重要となる汎用 CPU において特に問題となる．

本稿においては，コンピュータにおける参照の局所性を考慮し，実行時シグネチャのみならず，参照シグネチャもプログラム実行時に生成し，それらを比較することで制御フロー誤りを検出する手法，すなわち**動的シグネチャ検査 (DCSM: dynamic continuous signature monitoring)** 技術を提案する．あるプログラム部分の参照シグネチャと実行時シグネチャが一致すれば，制御フロー誤りはないものと期待でき，一致しなければ，制御フロー誤りがあると判断できる．当該プログラム部分が一度しか実行されない場合，一つの参照シグネチャが生成されるが，実行時シグネチャは生成されないため，二つのシグネチャを比較することができない．よってシグネチャ比較による制御フロー誤りを検出することができない．当該プログラム部分が二回以上実行される場合，一つの参照シグネチャと一つ以上の実行時シグネチャが生成されるために，この基本ブロックに対する制御フロー誤りは検出される．コンピュータでプログラムを実行する場合の多くは，いわゆる 90/10 ルール [3] が当てはまる．すなわち，一部のコードは複数回実行されるため，当該コード部分に対して複数回シグネチャを求めることができる．このような場合，当該コード部分に DCSM 技術を適用すれば，制御フロー誤りを検出することができる．本稿においては，DCSM 技術を提案するとともに，DCSM 技術によって保護される実行命令数について調査する．本稿の残りは次のように構成される．2. 節では既存の CSM 技術について概説する．3. 節では動的 CSM 技術を提案する．4. 節では，動的 CSM 技術によって達成される省メモリ性及び制御フロー誤り検出率を定量的に評価する．5. 節で本稿をまとめるとともに，本研究の今後の課題について述べる．

2. 制御フロー誤りと静的シグネチャ検査技術

マイクロプロセッサが正しく動作しているか否かを確認するためには，全てのサイクル時刻において回路中の全ての信号値を期待値と比較することが理想的である．しかしながら，数億に達する信号線を常時監視すること，及び，それらに対し期待値を用意することは非現実的である．マイクロプロセッサの動作に着目すると，ある命令列が正常に実行される場合，プログラムカウンタやデータパスの制御信号に代表される信号の系列はいつも同じものが生成される．これに対し，データ信号は命令列に対して一意に定まらない．制御信号のような一定の信号値の系列に対しては，一定の値に写像することが可能である．線形帰還シフトレジスタや加算器に代表される情報圧縮器を用いれば，制御信号の系列を情報圧縮し，ある値に写像すること

が可能である．本稿では，信号値の系列の写像をシグネチャと呼ぶ．制御信号の系列に誤りがあるか否かは，シグネチャをその期待値と比較することで確認することができる．本稿では，シグネチャの期待値を**参照シグネチャ(reference signature)** と呼ぶ．命令列が実行される度に計算される制御信号の系列に対するシグネチャを**実行時シグネチャ (runtime signature)** と呼ぶ．

典型的な CSM 技術においては，プログラムのコンパイル時などに制御信号の系列に対応する参照シグネチャを計算する．参照シグネチャはプログラムコード中に埋め込まれる場合が多い [7], [9], [17]．一方で，プログラムの実行中に，専用ハードウェアを用いて制御信号の系列に対して実行時シグネチャを求める．プログラム中に埋め込まれている参照シグネチャと，プログラム実行中に生成された実行時シグネチャを比較することで，制御フロー誤りの有無を確認できる．プログラムコード中にシグネチャ比較を行う命令を埋め込む既存手法は，メモリオーバヘッドと性能オーバヘッドを生じる．

以降，Wilken らによって一般化された CSM 技術概要について簡単に述べる [17]．連続して実行される命令列，すなわちパスと呼ばれる実行命令列を考えよう．長さが N であるパス上にある命令に対応づけられるシグネチャは，パスの各命令で行われる一連の中間的な計算の結果である．$i\,(1 \leqq i \leqq N)$ 番目の命令に対応付けられるシグネチャ S_i は以下のように求められる．

$$S_i = V(S_{i-1}, W_{i-1}), 1 \leqq i \leqq N \qquad (1)$$

ここで，V はシグネチャ関数，S_0 は初期シグネチャ，W_{i-1} は $i-1$ 番目の命令に対応づけられる信号値である．命令 i は中間シグネチャ S_i に対応づけられる．S_i は部分実行命令列 $[0, i-1]$ に対応するシグネチャである．

中間シグネチャは制御フロー誤りが検出可能かを決定する．正しい動作においては，プログラムは現在の命令 C から次の命令 D に遷移するとする．制御フロー誤りによって，命令 C から異なる命令 D^* への遷移を引き起こす．もし，$S_D = S_{D^*}$ であれば，命令 D^* から続くシグネチャ計算は命令 D^* を含むパスの参照シグネチャと一致する実行時シグネチャを生成し，その制御フロー誤りは検出できない．もし，$S_D \neq S_{D^*}$ であれば，命令 D^* を含むパスの参照シグネチャと異なる実行時シグネチャが生成されるために，その制御フロー誤りは検出可能である．

検出されない制御フロー誤りは中間シグネチャを用いて見積もることができる．各命令の中間シグネチャを決定し，シグネチャ S を持つ全ての命令の集合を D_S とする．D_S に属する命令が宛先である全ての命令の集合を C_S とする．制御フロー誤りは等確率で任意の命令にて生じ，等確率で任意の命令に誤って遷移するとする．d_S を D_S の大きさとし，c_S を C_S の大きさとする．m を全命令数とする．C_S に属する命令を起点とする，検出されない制御フロー誤りの割合は，$(d_S-1)/(m-1) \approx (d_S-1)/m$ (m が十分大きいため) となる．もし，全ての命令が均等に実行されるとすれば，C_S に属する命令で制御誤りが生じる割合は c_S/m となる．よって，検出されない制御誤り数の割合 e は，

$$e = \sum_i (d_S - 1) c_S / m^2 \qquad (2)$$

で表される．ただ，90/10 ルールを考えると，全ての命令は均等に実行されない．ここでは，CSM 技術の説明の簡便さのために「全ての命令が均等に実行される」と仮定する．

もし，中間シグネチャの値が命令に対して一様に分布しているとすれば，検出されない制御フロー誤りは最小となる．中間シグネチャ間の相関は，同じシグネチャを持つ命令の集合の大きさを大きくし，それゆえに，制御フロー誤りの検出率を低下する．CSM 技術のナイーブな実施例においては，初期シグネチャ S_0 に決まった値を用いる．これによって，全てのパスの最初の命令は D_{S_0} に属することになり，最後の命令は C_{S_0} に属することになる．パスの 2 番目の命令に対応する中間シグネチャ S_1 は，初期シグネチャ S_0 と信号値系列の始めの信号値 W_0 によって計算される値である．相関を持つ S_0 と W_0 を用いて計算された，異なるパスの S_1 同士は相関を持つこととなる．

検出されない制御フロー誤りの割合の下限は S_0 に起因する検出されない制御フロー誤りのみを考慮することで求められる．平均パス長 p に対し，D_{S_0} に属する命令の割合は $1/p$ であり，C_{S_0} に属する命令の割合は $1/p$ である．パス数が十分に多いとすると，シグネチャ S_0 に起因する検出されない制御フロー誤りの割合 e_{S_0} は，

$$e_{S_0} \approx p^{-2} \qquad (3)$$

となる．これまでの研究で，平均パス長は 4 から 10 であることが報告されている [2], [5]〜[7], [9]．図 1 に示すように，CSM を適用することにより，基本ブロックの末尾にシグネチャ命令が挿入されると仮定すると，平均パス長は 5〜11 となる．式 (3) より，検出されない制御フロー誤りの比率は 1〜4%となる．これは，96〜99%の制御フロー誤り検出率を意味する．v ビットのシグネチャを用いる場合，一様な乱数である中間シグネチャに対する制御フロー誤りの検出率は $1 - 2^{-v}$ で示される．$v = 16$ の場合，99.9985%の制御フロー誤りの検出率となる．既存の静的 CSM 技術は，シグネチャをメモリに保存するために，メモリオーバヘッドを避けるために，v の値を大きくできない．

図 2 に，制御フロー誤りのマルコフモデルを示す．全てのパスにおいて始めの命令の中間シグネチャ S_0 が共通である場合，第一義的には，式 (3) に示すように p^{-2} の確率で制御誤り

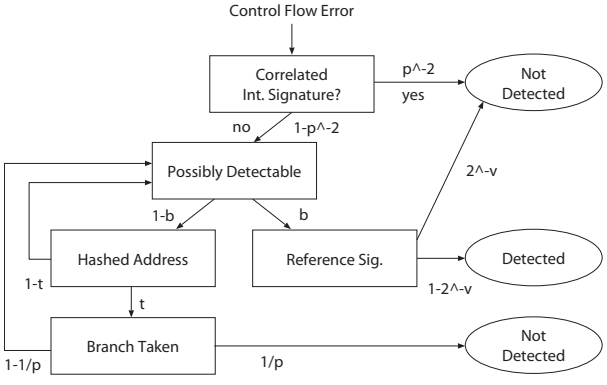

図 2　制御フロー誤りの算出．

を検出できない．残るエラーは検出できるかもしれない．参照シグネチャで終わるブロックの割合が b であるとする．ブロックの終わりがシグネチャ命令の場合，参照シグネチャと実行時シグネチャが比較される．制御フロー誤りによって，ある命令に実行が移り，その命令の実行時シグネチャは 2^v 通りの値になり得る．たまたま，その命令に対応づけられる実行時シグネチャになるかもしれず，その確率は 2^{-v} である．この制御フロー誤りは検出されない．その命令に対応づけられない実行時シグネチャになる確率は $1-2^{-v}$ であり，この場合制御フロー誤りは検出される．ブロックの最後の命令が分岐命令である割合は $1-b$ である．当該分岐命令には BHA (branch address hasing) [11] が適用されているとする．BHA は分岐先アドレスと reference address の排他的論理和をプログラムコード中に埋め込み，プログラム実行中に実行時シグネチャとハッシュ化されたアドレスの排他的論理和を計算し，分岐先アドレスを計算する手法である．誤った実行時シグネチャを用いて分岐先アドレスが計算されると誤ったアドレスに分岐することが期待できる．この分岐命令が not taken の場合，後続の命令の実行時シグネチャが誤ったものであることが期待されるために，制御フロー誤りが検出されることが期待される．分岐命令が taken の場合，誤っている実行時シグネチャで分岐先アドレスを計算する．分岐先アドレスがパスのブロックの先頭アドレスである場合，中間シグネチャは S_0 で初期化されるために，制御フロー誤りは検出されない．ブロックの先頭以外に分岐する場合，分岐先命令に対応づけられる中間シグネチャと初期化シグネチャ S_0 が異なることが期待されるので，後ほど実行されるシグネチャ命令で制御フロー誤りが検出されることが期待される．

3. 動的シグネチャ検査技術

本節では，実行時シグネチャと共に参照シグネチャもプログラム実行中に計算する**動的制御フロー誤り検出 (DCSM: dynamic CSM)** 技術の提案を行う．静的な CSM 技術においては，プログラムのコンパイル時に参照シグネチャを計算し，プログラムコード中などに埋め込む．併せて，参照シグネチャと実行時シグネチャを比較するための命令もプログラムコード中に埋め込む．対照的に DCSM 技術においては，基本ブロックのような，一定の信号値系列を得られるプログラム部分に対して，プログラムの実行中に参照シグネチャを計算し，それをシステム中の記憶装置に保存しておく．当該プログラム部分が再度実行される際に，実行時シグネチャを記憶装置中の参照シグネチャと比較することで，制御フロー誤りの有無を確認できる．DCSM 技術では，プログラムコード中に参照シグネチャ及びシグネチャ命令を埋め込む必要がないので，ハードウェア間の命令セットの互換性が保持される．また，DCSM 技術は静的 CSM 技術のようにメモリオーバヘッドを生じない．DCSM 技術では，当該プログラム部分に対応する参照シグネチャがすでに求められていれば，それと実行時シグネチャを比較する，さもなければ当該プログラム部分の実行時シグネチャを参照シグネチャとして記憶装置中に登録するといった機能をハードウェアで実現する．参照シグネチャの登録，及び参照シグネチャと実行時シグネチャの比較は自動的にハードウェアが行うため，参照シグネチャと実行時シグネチャを比較するシグネチャ命令を挿入する必要がない．しかしながら，シグネチャ生成の対象となるプログラムコード部分が一度しか実行されない場合，実行時シグネチャが生成されないので，このプログラムコード部分に関する制御フロー誤りを検出することができない．

図 3 に DCSM を行う回路の構成例を示す．DCSM の回路は大きくはシグネチャ生成器，パス解析器，およびシグネチャテーブルからなる．静的な CSM 技術においては，パス解析器及びシグネチャテーブルは存在しない．DCSM 技術を実現する上で，これらのハードウェアはチップ面積のオーバヘッドとなる．パス解析器はパスを検出する回路であり，プログラムカウンタの値及びプログラムカウンタが指し示す命令からパスを検出する．シグネチャテーブルはパスに対応する初期化シグネチャ及び参照シグネチャを保持する．なお，パスとは実行開始アドレスと実行終了アドレスの対で表現され，命令系列が唯一に定まるものとする．パス解析器がパスの始まりを検出した場合は，パスに対応するエントリが登録されているかシグネチャテーブルを検索する．シグネチャテーブルに当該パスのエントリが存在する場合，シグネチャテーブル上に登録されている当該パスの初期化シグネチャを用いてシグネチャ生成器を初期化する．さもなければ，式 (1) により得られるシグネチャを当該パスの初期化シグネチャとする．パスの終わりを検出した場合，当該パスのエントリがシグネチャテーブルに存在するか確認する．当該パスのエントリが存在する場合，シグネチャテーブル上に登録されている当該パスの参照シグネチャと実行時シグネチャを比較する．さもなければ，当該パスの初期化シグネチャ及び参照シグネチャをシグネチャテーブルに登録する．

DCSM 技術におけるパスの検出について考えてみよう．DCSM 技術においては，シグネチャ生成の対象となるパスを自動的に検出する必要があるが，静的な CSM 技術と異なり，高度にパスの切り分けを行うことができない．

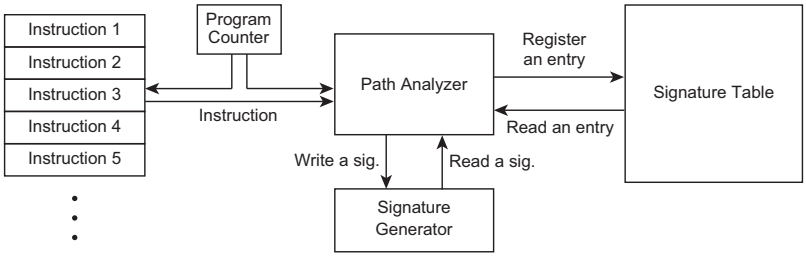

図 3　DCSM の概要.

例えば，静的な CSM 技術においては，プログラムコードに対するグローバルな最適化によってパスの切り分けを行い，シグネチャ比較回数やメモリ量を最小化できる．DCSM においては，このような高度な最適化は難しい．本稿では，プログラム実行中にパスを検出する点のみに焦点を当てる．パスを求める上で二つの分類を行う．一つは，命令アドレスが連続した命令系列をパスとする方法であり，もう一つはジャンプ命令・分岐命令による不連続な命令系列をパスとする方法である．前者を**第一種のパス**と呼び，後者を**第二種のパス**と呼ぶ．例えば，第一種のパスに対しては，以下の方針でパスを検出すれば良い．

- プログラム実行開始アドレスはパスの始めとみなす．
- ジャンプ命令及び分岐命令によって次に実行される命令が後続命令とならない場合，制御が移った命令をパスの始めとみなす．
- ジャンプ命令をパスの終わりとみなす，あるいは分岐が成立した命令をパスの終わりとみなす．遅延分岐スロットがある場合は，遅延スロット内にある命令の最後をパスの終わりとみなす．

また，第二種のパスに対しては，以下の方針でパスを検出すれば良い．

- ジャンプ命令・分岐命令によって連続しない命令アドレスに制御が移る場合，当該ジャンプ命令・分岐命令の命令アドレスをパスの始めとみなす．
- ジャンプ命令・分岐命令の宛先の命令アドレスをパスの終わりとみなす．

もちろん，上記のパス検出の方針は一例である．当該ジャンプ・分岐命令の前に出現するジャンプ・分岐命令以外の命令をパスに含めても良いし，当該ジャンプ命令・分岐命令の宛先の命令アドレス以降に出現するジャンプ・分岐命令以外の命令もパスに含めても良い．これらの命令をパスに含めても，「分岐が必ず起こる」という条件の下では「パスが一意に定まる」ために，当該パスはパスの要件を満たす．

4. 実　　験

本節では，計算機実験について述べ，考察を行う．4.1 節は，実験に用いたプログラムトレース，シミュレータ，及びシグネチャテーブルについて述べる．4.2 節では，評価に用いた尺度について述べると共に，実験結果を提示する．4.3 節で考察を行う．

4.1 実験準備

本実験においては，DCSM が適用された CPU の動作を模擬するトレース駆動シミュレータを開発し，評価を行った．開発言語は C++である．トレース駆動シミュレータの入力となるトレースは，Imperas 社の命令セットシミュレータ OVPsim により求めた．なお，OVPsim の実装上の制約により，システムコール部分のトレースは含めないこととした．今回の DCSM の実装形態としては，第一種のパスに対するシグネチャテーブル，及び第二種のパスに対するシグネチャテーブルの二つのシグネチャテーブルから構成されるハードウェア構成を採用した．シグネチャテーブルの記憶方式として，キャッシュメモリに用いられているセットアソシアティブ方式を採用した．シグネチャテーブルのライン置き換えアルゴリズムとして，LRU (least recently used) 方式を採用した．表 1 に示すセット数及びウェイ数に対して網羅的にトレース駆動シミュレーションを行った．対象としたベンチマークプログラム及びその実行命令数を表 2 に示す．

表 1　シグネチャテーブルのパラメータ．

セット数	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024
ウェイ数	1, 2, 4, 8, 16, 32, 64

表 2　ベンチマークプログラム．

ベンチマークプログラム	dhrystone	fibonacci	linpack	peakSpeed1
実行命令数 [10^6 命令]	1,012	747	1,704	3,440

4.2 実験結果

DCSM 技術で実現される信頼性尺度を与えるために，命令網羅率 C_{insts} を導入し，以下のように定義する．

$$C_{insts} = \frac{INST_{COVERED}}{INST_{ALL}} \times 100 \qquad (4)$$

ただし，$INST_{COVERED}$ は，シグネチャ比較により制御フロー誤り検出の対象となる実行命令数を，$INST_{ALL}$ は総実行命令数を示す．

表 3 に，計算機実験によって得られた，各プログラムにおける命令網羅率を示す．表 3 に示す命令網羅率は，表 1 に示すシグネチャテーブルのセット数・ウェイ数に対してトレース駆動シミュレーションを行った結果得たものである．表 3 に示すように，連続した命令系列部分の実行命令数が全体の実行命令数において大きな割合を占めることがわかる．ジャンプ・分岐命令の起因する制御フロー誤りの比率は小さいものの，10%を越える場合もあり無視できないことがわかる．

表 3　命令網羅率 C_{insts}．

	dhrystone	fibonacci	linpack	peakSpeed1
第一種パス	39.1-91.3	22.2-88.0	1.2-93.5	97.7-97.7
第二種パス	2.6-8.7	1.7-12.0	0.2-6.5	2.3-2.3
合計	41.8-100.0	23.9-100.0	1.4-100.0	100.0-100.0

図 4，及び 5 に，dhrystone における，シグネチャテーブルのセット数及びウェイ数に対する命令網羅率を示す．なお，他のプログラムについては，紙面 の都合のために割愛する．

4.3 考　　察

linpack や peakSpeed1 のように，特定の基本ブロックが高頻度に繰返し実行されるようなプログラムにおいては，DCSM 技術による命令網羅率は高くなる傾向にある．その処理内容から dhrystone や fibonacci は linpack や peakSpeed1 ほどには高頻度に特定の基本ブロックを実行しないと考えられ，dhrystone 及び fibonacci の最大命令網羅率はそれぞれ 91.3%及び 88.0%であることが確認できる．なお，Namjoo の手法 [7]，Shen らの手法 [9]，及び Wilken らの手法 [17] のいずれにおいても制御フロー誤り命令網羅率 C_{insts} は 100%となる．手法に応じて，制

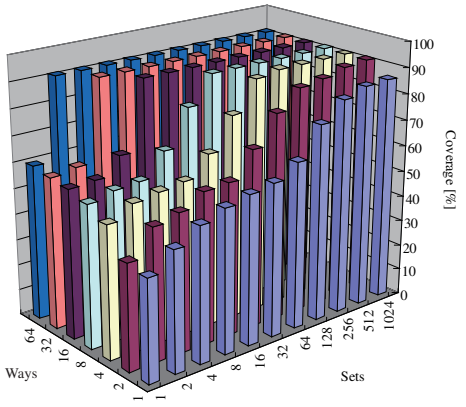

図 4　命令網羅率 (dhrystone, 連続した命令系列).

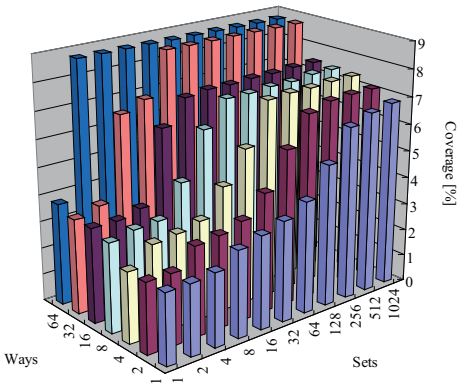

図 5　命令網羅率 (dhrystone, ジャンプ/分岐命令).

御フロー誤り命令網羅率よりも低い割合で制御フロー誤りが検出される.

コードの互換性を失っても良い場合，メモリオーバヘッドを許容できる場合，かつ性能低下を許容できる場合，DCSM による制御フロー誤り検出の対象とならない命令にのみシグネチャ命令を埋め込むことで，静的な CSM より少ないメモリ容量で同等の信頼性を実現できると考えられる.

5. おわりに

本稿では，動的制御フロー誤り検出 (DCSM: dynamic continuous signature monitoring) 技術を提案した．DCSM 技術は，参照シグネチャをプログラム実行時に求めることで，命令コードの互換性を保つと共に，メモリ量のオーバヘッドなしに信頼性を向上する技術である．本稿においては，DCSM 技術の実装の一例として，連続した命令系列部分，及びジャンプ・分岐命令に起因する不連続な命令系列部分のそれぞれに対して DCSM 技術を適用し，DCSM 機構の命令網羅率を測定した．最大 64K エントリのシグネチャテーブルを二つ用いれば，100%に近い命令網羅率を達成できることを示した.

本研究の今後の課題として，静的な CSM 技術と DCSM 技術を併せて用い，高信頼性及び省メモリ性を実現する技術，マルチコア CPU への応用，及び ASIC への応用が挙げられる.

謝　辞

本研究は，科学技術振興機構 (JST) の戦略的創造研究推進事業 (CREST) の研究領域「ディペンダブル VLSI システムの基盤技術」の支援の下に推進されました.

文　献

[1] V. D. Agrawal, C. R. Kime, and K. K. Saluja, “A tutorial on built-in self-test,” IEEE Design and Test of Computers, Vol. 10, Issue 1, pp. 73-82, March 1993.

[2] J. DeRosa and H. Levy, “An evaluation of branch architectures,” *Proc. IEEE 14th Comp. Arch.*, pp. 10-16, 1987.

[3] J. L. Hennessy and D. A. Patterson, Computer Architecture, p. 38, Morgan Kaufmann, 2007.

[4] Imperas Ltd., OVPsim and Imperas CpuManager user guide, 2009.

[5] M. Kobayashi, “Dynamic profile of instruction sequences for the IBM system/370,” *IEEE Transactions on Computers*, Vol. C-32, pp. 859-861, September 1983.

[6] A. Mahmood and E. McCluskey, “Watchdog processors: error coverage and overhead,” *Proc. IEEE 15th FTCS*, pp. 214-219, 1985.

[7] M. Namjoo, “Techniques for testing of VLSI processor operation,” *Proc. IEEE International Test Conference*, pp. 461-468, 1982.

[8] D. K. Schroder and J. A. Babcock, “Negative bias temparature instability: road to cross in deep submicron silicon semiconductor manufacturing,” *Journal of Applied Physics*, Vol. 94, pp. 1-18, July 2003.

[9] M. Schuette and J. Shen, “Processor control flow monitoring using signatured instruction streams,” *IEEE Transactions on Computers*, Vol. C-36, pp. 264-276, March 1987.

[10] N. Seifert, X. Zhu, D. Moyer, R. Mueller, R. Hokinson, N. Leland, M. Shade, and L. Massengill, “Frequency dependence of soft error rates for sub-micron CMOS technologies,” in the Technical Digest of *IEEE International Electron Devices Meeting (IEDM)*, pp. 14.4.1–14.4.4, December 2001.

[11] J. P. Shen, and M. A. Schuette, “On-line monitoring using signatured instruction streams,” *Proc. IEEE International Test Conference*, pp. 275–282, October 1983.

[12] M. Sugihara, T. Ishihara, and K. Murakami, “Task scheduling for reliable cache architectures of multiprocessor systems,” *Proc. IEEE/ACM Design, Automation and Test in Europe Conference*, pp. 1490-1495, Nice, France, April 2007.

[13] M. Sugihara, “SEU vulnerability of multiprocessor systems and task scheduling for heterogeneous multiprocessor systems,” *Proc. IEEE International Symposium on Quality Electronic Design*, pp. 757-762, San Jose, CA, USA, March 2008.

[14] M. Sugihara, T. Ishihara, and K. Murakami, “Reliable cache architectures and task scheduling for multiprocessor systems,” *IEICE Transactions on Electronics*, Vol. E91-C, No. 4, pp. 410-417, April 2008.

[15] M. Sugihara, “Reliability inherent in heterogeneous multiprocessor systems and task scheduling for ameliorating their reliability,” *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E92-A, No. 4, pp. 1121-1128, April 2009.

[16] M. Sugihara, “Heterogeneous multiprocessor synthesis under performance and reliability constraints,” *Proc. EUROMICRO Conference on Digital System Design*, pp. 333-340, Patras, Greece, August 2009.

[17] K. Wilken and J. P. Shen, “Continuous signature monitoring: low-cost concurrent detection of processor control errors,” *IEEE Transactions on Computer-Aided Design*, Vol. 9, No. 6, pp. 629-641, June 1990.