社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－EVA－18 (1)
2006／8／1

メモリ・アーキテクチャ・ベンチマーキング手法の提案

小野貴継† 井上弘士‡ 村上和彰‡

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‡

本稿では，高い精度を維持しつつ，短時間でのシミュレーションを可能とするメモリ・アーキテクチャ・ベンチマーキング手法を提案する．一般に，メモリ・アーキテクチャの評価では，アドレス・トレースに基づいたシミュレーションを行う．アプリケーション・プログラムの高機能化によりアドレス・トレースサイズが増加していることからシミュレーション時間が長くなる傾向にあり，シミュレーション時間の短縮が不可欠である．アドレス・トレースサイズを削減することでシミュレーション時間を短縮できるが，精度が低下するという問題がある．そこで本手法は，まず，トレースを小規模なトレースに分割し，それぞれの類似性に基づき代表となるトレースを選択する．これによりシミュレーションするトレースが小さくなり，時間を短縮できる．キャッシュ性能測定に基づく評価実験の結果，本手法はシミュレーション時間を平均 77.6%短縮し，そのときのキャッシュヒット率の予測誤差は平均 4.2%であった．

A Methodology for Memory Architecture Benchmarking

Takatsugu Ono† Koji Inoue‡ Kazuaki Murakami‡

Graduate school of Information Science and Electrical Engineering, Kyushu University†
Faculty of Information Science and Electrical Engineering, Kyushu University‡

In order to determine the memory architecture from a lot of design candidates, we use a trace-driven simulation. It is a common approach for evaluating memory architecture. However, it also demands much time. In this paper, we propose a Memory Architecture Benchmarking technique. It is possible that to reduce the simulation time while maintaining simulation accuracy. In order to evaluate validity of proposed technique, we measured the cache hit ratio. In our evaluation, the proposed technique reduces the simulation time about 77.6% and cache hit ratio prediction errors about 4.2% in the average.

1. はじめに

トランジスタの製造プロセスの進歩により，集積度が著しく向上しており，単位面積あたりのトランジスタ数が増加している．これにより，一つのチップ上に設計可能な回路が増加し，設計の選択肢が広がっている．本稿では特に，設計選択肢が多く性能への影響も大きい，メモリ・アーキテクチャに着目する．

メモリシステムの性能を評価する一つの方法としてシミュレータの利用が挙げられる．所望の機能を実装したメモリシステム・シミュレータを構築し，プログラム実行におけるメモリ・アクセス・トレースを入力する．この場合，1回あたりのシミュレーション時間はトレースサイズと共に増加する．したがって，実行対象となるアプリケーション・プログラムが大規模化するにつれ，シミュレーション時間はより長くなる．

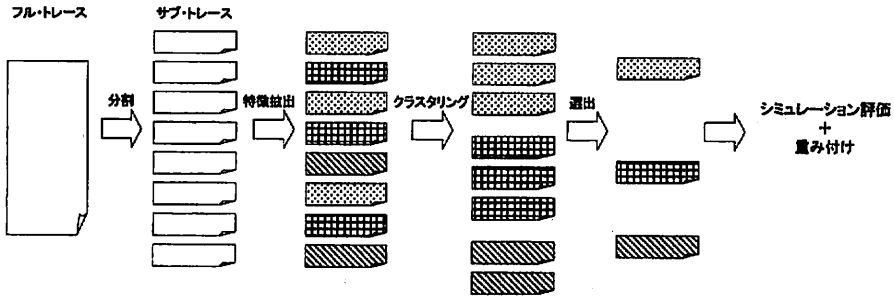


図1　メモリ・アクセス・ベンチマーキング手法の概要

この問題を解決する代表的な方法として，プログラムの部分実行による評価がある．例えば，プログラム実行開始からある一定数の命令を高速に実行した後，メモリ・アクセス・トレースを採取する．しかしながら，このようにして入手したトレースは，プログラムの全実行に関するメモリ参照の特性を必ずしも反映するとは限らない．そのため，シミュレーション結果の精度が低下するといった問題が生じることがある．

そこで本稿では，高い精度を維持しつつ，短時間でのシミュレーションを可能とするメモリ・アーキテクチャ・ベンチマーキング手法を提案する．また，実アプリケーションとしてMPEG2 を用いた定量的評価を行う．本手法は，まず，プログラムの実行開始から終了までを対象とした全メモリ・アクセス・トレースを取得する．そして，それを小規模メモリ・アクセス・トレースに分割し，それぞれの類似性に基づき代表となるトレースを選択する．これによりメモリ・アクセス・トレースが小さくなり，シミュレーション時間が短縮する．

以下，第 2 節では本稿で提案するメモリ・アーキテクチャ・ベンチマーキング手法について説明し，第 3 節では提案手法の有効性を調査する．第 4 節では関連研究について述べ，提案手法との違いについて説明する．最後に第 5 節で簡単にまとめ，今後の課題について述べる．

2.　メモリ・アーキテクチャ シミュレーション時間短縮

本節では，プロセッサとアプリケーション・プログラムが与えられているという前提における，メモリ・アーキテクチャのシミュレーション時間短縮法について述べる．

2.1　用語の定義

本稿で使用する用語について定義する．

- メモリ・アクセス：メモリへのアクセスが生じた際の時刻とアドレス
- メモリ・アクセス・トレース：メモリ・アクセスの集合
- フル・トレース：プログラムの実行開始から終了までのメモリ・アクセス・トレース
- サブ・トレース：フル・トレースの部分集合であり，互いに素である．

2.2　メモリ・アーキテクチャ・ベンチマーキング手法

シミュレーション時間はアドレス・トレースサイズを削減することで短縮できるが，シミュレーション精度が低下することが予想される．なぜなら，削減されたメモリ・アクセスが性能に与える影響をシミュレーションできないからである．

そこで本稿では，シミュレーション精度を維持しつつ，トレースサイズを削減するメモリ・アーキテクチャ・ベンチマーキング手法を提案する．図 1 に提案手法の概要を示す．まず，フル・トレースを一定時間でサブ・トレースに分割し，各サブ・トレースにおける特徴を抽出する．同じ特徴を持つサブ・トレースは，シミュレーション結果も同じと考えることができるため，そのうち一つだけをシミュレーションすれば良い．そこで，抽出した特徴に基づいてサブ・トレースをクラスタリングし，各クラスタから代表サブ・トレースを一つずつ選出する．最後に，選出した各サブ・トレースを用いてシミュレーションし，結果に重み付けする．

2.3 メモリ・アクセスの特徴抽出

サブ・トレースにおけるメモリ・アクセスの特徴抽出法について説明する．縦軸をアドレス，横軸を時間として，サブ・トレースにおけるメモリ・アクセスを直交座標上にプロットしたグラフを考える．これに対して，以下の手順で特徴を抽出する．

1. 図 2 のように，アドレス軸方向に等間隔で分割する．以後この間隔のことを時間インターバルという．時間軸方向に対しても同様に等間隔で分割する．以降この間隔のことをアドレスインターバルという．また，時間インターバルおよびアドレスインターバルで囲まれた範囲をブロックと呼ぶ．尚，図 2 における点はメモリ・アクセスを表している．
2. アドレス軸方向のブロック数 n，時間軸方向のブロック数 m とすると，n 行 m 列の 2 次元配列を用意する．図 2 の例においては，アドレス軸方向のブロック数が 6，時間軸方向の数は 4 であるため，6 行 4 列の配列を用意する．各ブロックにおけるメモリ・アクセス数を，配列の各要素とする．例えば図 2 において，左上のブロックには一つのメモリ・アクセスがある．したがって，作成した配列の 1 行 1 列の要素は 1 となる．他のブロックについても同様の処理を行う．
3. 各時間インターバルの原点を 0 として，メモリ・アクセスの時間軸方向の原点からの距離を求める．この距離の平均値と標準偏差値を求める．n×m 行列に 2 行新たに追加し，第 n+1 行の要素を平均値，第 n+2 行の要素を標準偏差値とする．さらに，各時間インターバル内のメモリ・アクセスにおいて，直前のメモリ・アクセスとのドレスの差を求め，その差の絶対値の総和を算出する．以後これをアドレス間距離という．そして(n+2)×m 行列に，新たに 1 行追加する．追加した第 n+3 行の要素をアドレス間距離とする．図 3 において第 6 行までのブロックと示している部分は手順 1 で用意した配列である．第 7 行から第 9 行の 3 行がこの手順 3 で追加した部分である．
4. (n+3)×m 行列の列を任意の値に置き換えることによって，m 次元のベクトルに変換する．まず，各列のそれぞれの要素を比較し，一致する列に同じ値を付ける．例えば，図 4 において，第 1 列と第 4 列の各要素は一致するため同じ値，ここでは“1”を付ける．第 2 列と第 3 列はどの列にも一致しないため，他の列とは異なる番号を付ける．図 4 の例では“2”，“3”とした．ただし，各要素の値の差が 10%までは一致とみなす．

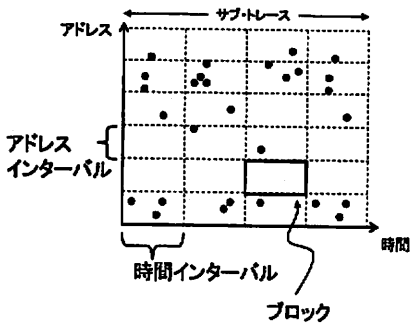


図 2　サブ・トレースの分割

ブロック	1	0	1	1
	2	4	3	2
	1	1	0	1
	0	1	1	0
	0	0	0	0
	3	2	0	3
平均値	4.5	4.8	6.2	4.5
標準偏差値	1.1	2.3	2.3	1.1
アドレス間距離	55.5	33.5	25	55.5

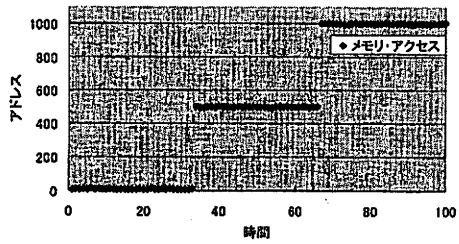
図 3　特徴を表す 2 次元配列

1	0	1	1
2	4	3	2
1	1	0	1
0	1	1	0
0	0	0	0
3	2	0	3
4.5	4.8	6.2	4.5
1.1	2.3	2.3	1.1
55.5	33.5	25	55.5

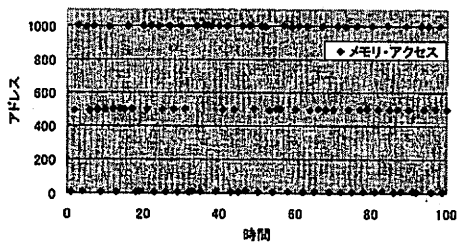
⇩ ⇩ ⇩ ⇩

1	2	3	1

図 4　2 次元配列を特徴ベクトルに変換



(a) 連続したメモリ・アクセス



(b) 離散的なメモリ・アクセス

図5 アドレス間距離の必要性

このようにして得られたベクトルのことを，特徴ベクトルという．

これらの手順をすべてのサブ・トレースに対して行う．

ブロック内のメモリ・アクセス数だけでは，時間インターバルにおけるメモリ・アクセスの時間的な偏りを表すことができない．そこで，時間的な偏りを表す指標として，手順 3 において平均値と標準偏差値を追加した．また，同じ手順 3 で追加したアドレス間距離は，メモリ・アクセスがあるアドレス付近に連続して行われているか否かを表す指標である．例えば，あるアドレス付近に時間的に連続してアクセスが生じた場合は値が小さくなる．一方，異なるアドレスに離散的にアクセスが生じた場合は値が大きくなる．図 5 の例を用いて具体的に説明する．図 5 は軸をアドレス，横軸を時間としてメモリ・アクセスをプロットしたものである．図 5(a)と図 5(b)の，各ブロックにおけるメモリ・アクセス数と，時間インターバルにおける平均値及び標準偏差値は同じである．しかしながら，特徴が異なっていることが分かる．図 5(a)はあるアドレス付近に，時間的に連続してアクセスするのに対し，図 5(b)は離散的にアクセスしている．このとき，図 5(a)のアドレス間距離は 1,196 であり，図 5(b)は 59,918 である．これらの指標を追加することで，ブロック内のメモリ・アクセス数だけでは表すことのできない特徴を補う．

以降のメモリ・アクセス・ベンチマーキング手法の過程において，膨大なサブ・トレースをクラスタリングする．そのため，クラスタリング処理に要する時間が長くなると予想される．従って，クラスタリング処理時間を短縮するために各サブ・トレースの特徴を少ない情報で表現する必要がる．この理由から，手順 4 において(n+3)×m 行列をベクトルに変換した．

2.4 代表サブ・トレースの選出と重み付け

2.2 節で述べたように，サブ・トレースの特徴が同じ場合，それを用いたシミュレーション結果もまた同じであると考えられるため，全てをシミュレーションする必要はない．そこで，各サブ・トレースを特徴ベクトルに基づいてクラスタリングし，それぞれのクラスタから代表を一つ選出する．選出したサブ・トレースを用いてシミュレーションする．

クラスタに属するサブ・トレース数が多い場合は，フル・トレースを用いたシミュレーション結果に大きな影響を与えると考えられる．一方，少ない場合は結果に与える影響は小さいと考えられる．従って，クラスタに属するサブ・トレース数を用いて，各代表サブ・トレースを用いたシミュレーション結果に重み付けする．

クラスタ数はシミュレーション時間と精度に深く関わっている．クラスタ数が少なければ代表サブ・トレースが少ないためシミュレーション時間は短くなる．しかしながら，異なる特徴であっても同じクラスタに分類される可能性があるたるため，シミュレーション精度の低下を招く原因となる．一方，クラスタ数が多ければシミュレーション時間が長くなるが，その精度は高くなると考えられる．

3. 評価

提案手法の有効性を調査するために，キャッシュヒット率の測定を行う．フル・トレースを用いたシミュレーション結果と提案手法によって得られた結果とを比較する．そして，シミュレーション時間の削減率を求める．

3.1 実験環境

フル・トレースを取得するために，SimpleScalar[4]というマイクロプロセッサ・シミュレータ上で MPEG2 のデコードプログラム[3]を実行する．デコードする動画像として，mei16v2 と missa を使用した．命令及びデータキャッシュメモリを対象とし，アドレスインターバルと時間インターバルは共に 1,000 とした．また，一つのサブ・トレースは 10,000 クロック・サイクルとする．

クラスタリングのアルゴリズムは K - 平均法[12]を用いた．K - 平均法は短時間で実行できる上に，良いクラスタリング結果を得ることができる．K - 平均法によるクラスタリングが実行できるソフトウエアである Cluster3.0[1]を使用し，クラスタ数 K は 500，1,000，2,000 の三つとした．

各サブ・トレースの直前までキャッシュのウォームアップが完了しているという前提で行う．そこで，代表サブ・トレースの選出は以下の手順で行った．

1. フル・トレースを用いてシミュレーションし，キャッシュヒット率を測定する．このとき，サブ・トレースの間隔である 10,000 クロック・サイクルごとにキャッシュヒット率を求める．これらと各クラスタとを対応付けておく．
2. 特徴ベクトルに基づいてサブ・トレースをクラスタリングする．各クラスタにおけるサブ・トレースのキャッシュヒット率の平均値を求める．
3. 手順 2 で求めた平均値に最も近いキャッシュヒット率に対応するサブ・トレースを一つ選出する．

3.2 実験結果

提案手法によるフル・トレースサイズ削減率を示す．また，フル・トレースによるシミュレーション結果との予測誤差を示す．

3.3.1 フル・トレースサイズ削減率

命令キャッシュを対象とした，提案手法によるフル・トレースサイズの削減率を図 6 に示す．縦軸はトレースサイズの削減率，横軸は入力データであり，K はクラスタ数である．データキャッシュを対象としたフル・トレースサイズ削減率を図 7 に示す．縦軸及び横軸共に図 6 と同様である．

平均で 77.6%のフル・トレースサイズの削減を達成した．シミュレーション時間はメモリ・アクセス・トレースサイズに比例するため，シミュレーション時間もまた 77.6%程度削減できると考えられる．

フル・トレースサイズの削減率はクラスタ数によって決まる．クラスタ数が多い程，削減率は低い．missa における削減率が，mei16v2 よりも低い理由は，missa が，mei16v2 と比較してフル・トレースサイズが小さいからである．

3.3.2 キャッシュヒット率予測誤差

命令キャッシュを対象とした，提案手法によるキャッシュヒット率予測誤差を図 8 に示す．縦軸はキャッシュヒット率予測誤差，横軸は入力データであり，K はクラスタ数である．データキャッシュを対象としたキャッシュヒット率の予測誤差を図 9 に示す．縦軸及び横軸共に図 8 と同様である．

2.2 節で述べた様に，サブ・トレースサイズが大きいほどキャッシュヒット率の予測誤差は小さくなると考えられる．しかしながら，これらの結果は必ずしもそうとは限らない．この原因は主に二つある．

一つ目は，選出されたサブ・トレースのヒット率とクラスタの平均キャッシュヒット率との誤差が影響している場合である．特に，クラスタに属するサブ・トレースの数が多い場合は，その分大きな重みを付けるため誤差が大きく影響する．

二つ目に，その誤差の符号の影響が挙げられる．図 9 の mei16v2 の，K=500 および K=2,000 を例に説明する．これらは，フル・トレースサイズ削減率が 90%以上と高いにも関わらず，予測誤差はわずか 1%未満である．図 10 に図 9 の mei16v2 における予測誤差の内訳を示す．縦軸はキャッシュヒット率予測誤差の絶対値であり，横軸はクラスタ数である．それぞれの棒グラフは，本来のキャッシュヒット率よりも高く予測した結果と，逆に低く予測した結果の内訳を示している．図 10 より，K=500 と K=2,000 において，高く予測した結果と低く予測した結果の割合がほぼ等しいことがわかる．本来のキャッシュヒット率よりも高く，もしくは低く予測した結果同士が互いに打ち消しあう現象が生じている．結果的に，フル・トレースサイズの削減率が高い場合でも，キャッシュヒット率の予測誤差が小さくなっていると考えられる．また，2.4 節において，クラスタ数が多い程シミュレーション精度が高いという仮説を立てた．しかしながら，図 10 の結果はクラスタ数が多くなるにつれて予測誤差の絶対値が大

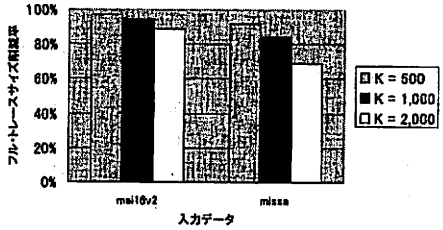


図6 命令キャッシュにおける
フル・トレースサイズ削減率

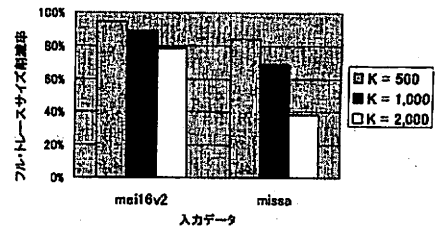


図7 データキャッシュにおける
フル・トレースサイズ削減率

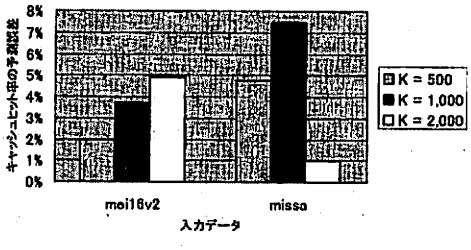


図8 命令キャッシュにおけるキャッシュ
ヒット率の予測誤差

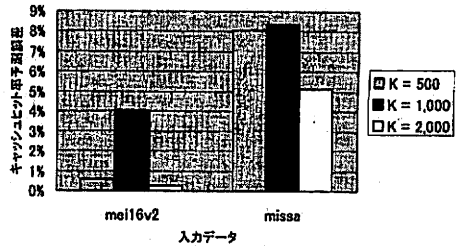


図9 データキャッシュにおける
キャッシュヒット率の予測誤差

きくなっている．この原因は先に述べた一つ目の理由と関係がある．クラスタ数が多い場合，それぞれに属するサブ・トレースの数が少なくなる．従って，クラスタの平均ヒット率に最も近いヒット率を有するサブ・トレースを選出する際に，誤差が生じやすくなる．

本来，提案手法は一度フル・トレースから代表サブ・トレースを選出すると，異なるキャッシュ構成についてもそれらを用いてシミュレーションすることを目的としている．しかしながら，本稿における評価では，キャッシュヒット率というキャッシュ構成に依存した指標によって代表を選出している．そのため，異なるキャッシュ構成において，その代表が必ずしも適切な代表であるとは限らない．今後，クラスタの中心を代表として選出するなどキャッシュ構成に依存しない異なる指標を用いる必要がる．

4. 関連研究

シミュレーション時間短縮を目的とした研究は多く行われている[11]．プログラムの特徴的な箇所を部分的に実行してシミュレーション時間を短縮するサンプリング手法がいくつか提案されている[2][5][6][9]．

SMARTS[9]は，ある固定インターバルによって命令ストリームをサンプリングする．このサンプリング周期とインターバルによってシミュレーション時間が決まる．各インターバルは，1) 詳細をシミュレーションし結果を測定，2) 詳細をシミュレーションするが結果は測定しない，3) 詳細なシミュレーションはせず機能シミュレーションのみを行う，の三種類に分類できる．3) の機能シミュレーションでは，キャッシュや分岐予測器の状態を継続的にアップデートしているが，詳細なシミュレーションに比べて時間は短い．SMARTS を用いた場合，CPI の平均誤差は極めて低いにも関わらず，35 倍から 60 倍のシミュレーション速度の向上を達成できる．しかしながら，メモリ性能のシミュレーション精度については言及していない．また，SMARTS のように一定周期でサンプリングする手法は，その周期にあわせて性能に大きな影響を与えるメモリ・アクセスなどが生じた場合，そのシミュレーション精度は低くなる．

SimPoint[5][6]はプログラムの実行から終了までを一定の命令数でインターバルに区切り，各インターバルにおける基本ブロックの実行回数に着目し特徴付けしている．この特徴をもとにクラスタリングし，一つのインターバルをシ

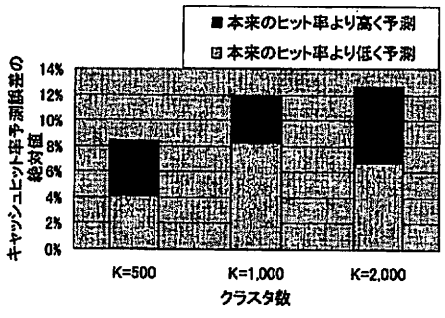


図 10　キャッシュヒット率予測誤差の内訳

ミュレーション・ポイントとして決定する．決定されたポイントのみをシミュレーションし，その結果に重み付けしている．また，基本ブロックではなくループ分岐によって特徴付けている研究[8]や，インターバルを可変にしている研究[7]もある．

SimPoint はアプリケーション・プログラム中の基本ブロックに着目しているのに対し，我々の提案手法はメモリ・アクセスの事象そのものに着目している．この違いから，本手法はより効率よく所望のアーキテクチャを探索することが可能である．例えば，過去に同じ特徴を持つサブ・トレースでシミュレーションしたとする．異なるアプリケーション・プログラムや入力データでも，特徴さえ同じであればシミュレーション結果は近いと予想できる．事前に過去のシミュレーション結果などから，メモリ・アクセスの特徴とメモリ・アーキテクチャとを関係付けておく．こうすることで，ある特徴に対してどのようなメモリ・アーキテクチャが適しているか，シミュレーションすることなく検討することができる．

また，キャッシュシの性能測定において，メモリ・アクセスの順序は重要である．なぜなら，順序次第では，本来キャッシュミスとなるメモリ・アクセスがヒット，逆にヒットとなるメモリ・アクセスがミスとなる可能性があるからである．本手法における特徴抽出ではメモリ・アクセスの順序も考慮している．しかしながら，SimPoint は基本ブロックの実行回数のみに着目しているため，メモリ・アクセスの順序は考慮していない．よって，キャッシュの性能測定を目的としたシミュレーションにおいて，SimPoint より提案手法のほうが，シミュレーション精度が高いと言える．

5.　おわりに

本稿ではメモリ・アーキテクチャのシミュレーション時間の短縮を目的とした，メモリ・アーキテクチャ・ベンチマーキング手法を提案した．また，提案手法の有効性を調査するため，MPEG2 のデコードプログラムを対象とした，キャッシュの性能測定に基づく評価実験を行った．MPEG2 のメモリ・アクセス・トレースの特徴を抽出し，この特徴に基づいてサブ・トレースを選出した．これらのサブ・トレースを用いてシミュレーションを行ったところ，シミュレーション時間の削減率は平均 77.6%に対し，キャッシュヒット率の測定誤差は平均 4.2%であった．

本稿の評価実験において，各代表サブ・トレースによるシミュレーションを行う前に，キャッシュのウォームアップが完了しているという前提で行った．今後，ウォームアップに必要なメモリ・アクセス・トレースサイズを考慮して評価を行う．

また，サブ・トレースの間隔と，時間及びアドレスインターバルの値は，特徴抽出結果に大きな影響を与えると考えられる．本稿においては，サブ・トレース間隔を 10,000 クロック・サイクル，時間及びアドレスインターバルを 1,000 としたが，異なる値でも評価する必要がある．

本稿ではアプリケーション・プログラムとして，MPEG2 を使用した．MPEG2 はデータ・ストリームに対して決まった処理を繰り返すため，メモリ・アクセスの特徴がある一定の周期で変化する．従って，異なる特徴を持つサブ・トレースが比較的少なかったと考えられる．今後は，メモリ・アクセスが不規則に生じるアプリケーション・プログラムについても，本手法の有効性を調査する予定である．

謝辞

本論文をまとめるにあたり，共にご議論頂いた九州大学システム LSI 研究センターならびに安浦・村上・松永・井上研究室の皆様に感謝します．なお，本研究は一部，科学研究費補助金(学術創成研究費:課題番号 14GS0218,若手研究A:課題番号 17680005)，21 世紀 COE プログラム「システム情報科学での社会基盤システム形成」，ならびに，松下電器産業株式会社との協同研究による．

参 考 文 献

[1] "Cluster3.0," URL : http://bonsai.ims.u-tokyo.ac.jp/~mdehoon/software/cluster/software.htm#source

[2] R. Kessler, M. Hill, and D. Wood, "A Comparison of Trace Sampling Techniques for Multi-Megabyte Caches," IEEE Transactions on Computers, vol. 43, no. 6, pp. 664-675, June 1994

[3] "MPEG Software Simulation Group," URL : http://www.mpeg.org/MPEG/MSSG/

[4] "SimpleScalar Simulation Tools for Microprocessor and System Evaluation," URL : http://www.simplescalar.org/.

[5] T. Sherwood, E. Perelman, G. Hamerly and B. Calder, "Automatically Characterizing Large Scale Program Behavior," International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 45-57, October. 2002.

[6] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Basic Block Distribution Analysis to Find Periodic Behavior and Simulation Points in Applications," International Conference on Parallel Architectures and Compilation Techniques, pp. 3-14, September. 2001.

[7] J. Lau, E. Perelman, G. Hamerly, T. Sherwood, and B. Calder, "Motivation for Variable Length Intervals and Hierarchical Phase Behavior," International Symposium on Performance Analysis of Systems and Software, Mar. 2005.

[8] J. Lau, S. Schoenmackers, and B. Calder, "Structures for Phase Classification," International Symposium on Performance Analysis of Systems and Software, Mar. 2004.

[9] R. Wunderlich, T. Wenisch, B. Falsafi, and J. Hoe, "SMARTS: Accelerating Microarchitecture Simulation via Rigorous Statistical Sampling," International Symposium on Computer Architecture, pp. 84-97, June. 2003.

[10] R. Uhlig and T. Mudge, "Trace-driven Memory Simulation: A Survey," ACM Computing Surveys, vol. 29, no. 2, pp. 128-170, 1997.

[11] J. Yi, and David. Lilja, "Simulation of Computer Architectures:Simulators, Benchmarks, Methodologies, and Recommendations," Transactions on Computers, Vol. 55, No. 3, pp. 268-280 March. 2006.

[12] 麻生英樹，津田宏治，村田昇 "パターン認識と学習の統計学　新しい概念と手法," 岩波書店