論 文 の 内 容 の 要 旨

論文題目　帰納的なシミュレーション・ポイント選出手法に関する研究
(A Study on Inductive Methods to Select Simulation Points)

氏　名　崔　珉誠

プロセッサのシミュレーションは，アーキテクチャ研究・開発の基礎であるが，実行に極めて長い時間がかかるという問題点がある．その理由は，以下の2つに分けられる．

第1 に，シミュレータの実行速度の遅さが挙げられる．

第2 に，評価に用いるベンチマーク・プログラムの命令数が非常に多いことが挙げられる．

その上，性能評価のためには通常，何十通りもパラメータを変えてシミュレーションを行う必要がある．

したがって，シミュレータ自体を高速化することは，重要なことではあるが，この文脈では根本的な解決にはならない．たとえ数倍程度の高速化が可能であったとしても，年単位のものが月単位に短縮されるだけで，評価に用いるには依然として非実用的であるからである．

そこで実際には，ベンチマーク・プログラムのごく一部のみをシミュレーションするサンプリング・シミュレーションが行われる．典型的には，プログラムの最初の数G 命令程度をエミュレーションによってスキップし，その後の数百M 命令程度のみをシミュレーションすることが多い．

このような方法は，広く用いられてはいるものの，シミュレーションした部分がベンチマーク・プログラムの特徴を確かに反映したものであるかどうか疑問が残る．

そのため，「よりよい」シミュレーション・ポイントを選出することが考えられる．シミュレーション・ポイントとは，そこだけを実行することによってプログラム実行全長にわたるターゲット・プロセッサの振る舞いが推定できるようなプログラム実行の一部のことである．

よりよいシミュレーション・ポイントを選出するには，フェーズ検出の考え方を用いればよい．

プログラムの繰り返し構造に起因して，プログラムの動的な区間の中にはプロセッサが同様の振る舞いを示すものが多くある．区間のうち，同様な振る舞いを示す区間は同じフェーズ，異なる振る舞いを示す区間は異なるフェーズと呼ぶことができる．ここで，同様な振る舞いとは，合目的的に定義すれば，IPC，および，その他の性能指標がお互いに近い値を示すことを意味する．

このような考え方に基づけば，プログラム実行の全区間を，例えば数十～数百種類のフェーズに分類し，同じフェーズに属する区間から1つずつを選んでシミュレーション・ポイントとすればよい．

フェーズ検出に基づいてシミュレーション・ポイントを選出する代表的な手法としてSimPoint が挙げられる．SimPoint をはじめとするほとんどの手法の特徴は，フェーズ検出（とシミュレーション・ポイントの選出）にプログラム・カウンタ(PC) の系列を用いることにある．

SimPoint は，まずターゲット・プログラムのエミュレーションを行い，（プログラム実行の全長にわたる）PC の系列を得る．そして，そのPC 列を固定長のインターバルに分割する．それぞれのインターバル

に含まれるPC の種類を基に，クラスタリングを行う．すなわち，もし，2 つのインターバルがほぼ同じPC を含む場合，それらは同じクラスタに分類されることになる．最後に，それぞれのクラスタの代表点をシミュレーション・ポイントの1 つとして選出する．IPC，および，その他の性能指標は，選出された各シミュレーション・ポイントの性能指標を，各クラスタのインターバルの数で重み付けした平均として推定される．

SimPoint をはじめ，PC を基礎とする手法は，以下を仮定していることになる；すなわち，プログラムの同じ静的部分を実行する動的部分は同じフェーズであり，ターゲット・プロセッサはこれらの部分で同様に振る舞う．

しかしこの仮定には明らかな反例がある．たとえ同じコードであっても，入力が異なれば，プロセッサが同じ振る舞いを示すとは限らない．典型的には，処理されるデータの量が異なれば，キャッシュ・ヒット率は大きく異なり，IPC は大きく変化し得る．実際，SPEC 2006 の一部のベンチマークでは，同じ部分が異なるサイズのデータに対して繰り返し実行されており，SimPoint の精度を大きく悪化させている．

そこで本研究では，特徴的なマイクロアーキテクチャを持ついくつかのプロセッサのモデルによってプログラムのシミュレーションを行い，その結果から汎用的なシミュレーション・ポイントを選出する手法を提案する．SimPoint などの既存手法は演繹的であるのに対して，提案された方法は帰納的であると言える．

提案手法では，先ず，シミュレーション対象プログラムをいくつかの特徴的なマイクロアーキテクチャを持つモデルでシミュレーションする．これらのモデルを基底モデル(basis model)と呼ぶ．基底モデルは互いに十分異なり，特徴的なIPC の振る舞いを示すものが望ましい．

基底モデルによる事前シミュレーションによって，インターバルごとのIPC ベクトルの系列を得る．このIPC ベクトルにおいて，

n 種のモデル M_1, M_2・・・M_n で，2 つのインターバルを事前シミュレーションすることで，2 個の n 次元の IPC ベクトル V_1 = (I_11,I_12, ・・・I_1n) と V_2 = (I_21,I_22,・・・I_2n) を得る．ここで I_ij はインターバル I におけるモデル M_j の IPC を表す．

これらに対して，V_1 ≃ V_2，すなわち，I_11≃I_21, I_12≃I_22,・・・I_1n≃I_2n が成り立つとする．M_1,M_2,・・・M_n のマイクロアーキテクチャが互いに十分異なっていれば，2 つのインターバルは同じフェーズであり，未知のモデル M_n+1 に対しても I_1n+1≃I_2n+1 となることが期待できる．そして，シミュレーションによって I_1n+1 だけを得ることによって，I_2n+1(≃ I_1n+1)を推定することができる．

次に，得られたIPC ベクトルに対しクラスタリングを行う．すなわち，距離の近いIPC ベクトルを同じフェーズに，離れているものを別のフェーズに分類する．

今回の評価では，k-means 法ではなく，以下のような単純なアルゴリズムを用いてクラスタリングを行った．あるIPC ベクトルv に対して：

1. 既にある全てのクラスタに対し，それらの重心とv との距離(通常，ユークリッド距離)を比較する．
2. v からの距離が閾値以内であれば，v を最も近いクラスタに追加する．重心を再計算しておく．
3. 閾値以内にクラスタがなければ，新しいクラスタを作成し，そのクラスタの最初のメンバとしてv を追加する．

すべてのインターバルに対して，1 から3 を繰り返す．

このクラスタリング方法では，閾値の大小が，シミュレーション・ポイントの量と推定精度との間のトレードオフを決定する．閾値は，ベクトル空間内の各クラスタの最大の「半径」を与える．閾値を小さくすると，より小さいクラスタが数多く生じる．その結果，より多くのシミュレーション・ポイントが選出され，推定精度が向上することになる．

クラスタリングの後は，SimPoint と同様である．シミュレーション・ポイントの選出では，それぞれクラスタの重心に最も近いインターバルを取り出し，これをシミュレーション・ポイントの1 つとする．IPC，および，その他の性能評価指標の推定では，選出されたシミュレーション・ポイントのみをシミュレートし，各クラスタのインターバルの数で重み付けした平均をもって，プログラム全体に対する評価指標を推定することができる．

また，基底モデルの選定において，

現代のスーパースカラ・プロセッサにおけるIPC 変動の主な要因は以下① キャッシュ・ミス ② 分岐予測ミス ③ 命令の依存関係 の3つである．

プロセッサの研究・開発ではこれらの影響を減らして性能を向上（あるいは維持）させることに主眼が置かれている．そこで，初期の評価に際しては，基底モデルは以下の4 つとした：

superscalar 基本構成である一般的な4-way out-of-order スーパースカラ・プロセッサ．

scalar キャッシュを持つスカラ・プロセッサ．

cache-perfect キャッシュのヒット率を100%にしたスーパースカラ・プロセッサ．

bpred-perfect 分岐予測のヒット率を100%にしたスーパースカラ・プロセッサ．

最初の3 つは，いずれもIPC 変動の要因のうち，いくつかの影響を全く受けない，「理想的な」マイクロアーキテクチャである．これらの「理想的な」モデルは，性能の上限を与え，これら以上に特徴的なIPC の変動はないと考えられる．

また，これら3つの理想的なマイクロアーキテクチャに対する比較の意味で，一般的なスーパースカラ・プロセッサを基底モデルに追加している．現在ある，あるいは，将来開発される「現実的な」プロセッサの性能はこれらの「理想的な」マイクロアーキテクチャと一般的なスーパースカラ・プロセッサの「間」にあると考えられる．すなわち，これらのモデルは，「現実的な」モデルを表現するための基底ベクトル（basis vector）の役割を果たすと言える．

さらに，いくつかの帰納的な選出手法の改良を行う．

第1 に，キャッシュ容量固有のフェーズに対応する多階層キャッシュを基底モデルに追加する．

キャッシュは様々な容量を取ることが可能であり，容量固有のフェーズが生じる可能性がある．そこで，様々な容量のキャッシュからなる，多階層キャッシュを持ったアーキテクチャ（多階層キャッシュ）を基準アーキテクチャに加える．多階層キャッシュは，ワーキングサイズが徐々に大きくなる区間においては，図のように徐々にIPC が低下していく．よって，一定の性能差おきに別々のフェーズとして分類できるので，様々なキャッシュ容量に対応してフェーズがとれる．

第2 に，提案手法では，シミュレーション・ポイントを選出するためにいくつかの基底モデルを事前シミュレーションする．事前シミュレーションは，原則的には，ベンチマークに対して一度だけ実行すればよい．それでもなお，事前シミュレーションにかかる時間は膨大である．そこで，IPC 変動の主な要因からの理想的なモデルをエミュレーション・ベースの基底モデルを使って，シミュレーション・ポイントを

選出してIPC を推定することで，時間短縮を果たす．

プログラム中には分岐があると，分岐先の予測を行い，予測結果に基づいて次の命令アドレスを決定する．一方，予測が誤った分岐があれば，分岐予測ミス・ペナルティが発生する．そこで，分岐予測モデルは分岐予測ミス・ペナルティを追跡することによって分岐予測だけのIPC の遷移を取り出す．

もう一つ，IPC 変動の主な要因は命令間の依存関係である．命令の間に依存があれば，それらの命令は逐次に実行されなければならない．すなわち，命令レベル並列性(Instruction-Level Parallelism)である．

命令依存性のみがIPC 変動に反映されるようなモデルとして，無限のハードウェアリソースを持つ．ただし，インターバル単位でのILP を反映するために，インターバル幅の命令ウィンドウを持つとする．このようなモデルでは，インターバル内での命令依存にない命令はすべて並列に同時実行されるため，IPC 変動には命令依存性のみが反映される．

最後に，Out-of-order プロセッサでは，MLP (Memory-Level Parallelism) を抽出することによって，すなわち，キャッシュ・ミスの処理中に後続の命令を実行することによって，区間によってはin-order プロセッサに比べて何十倍もの高速化を達成することがある．この効果は，キャッシュと命令レベル並列性の組み合わせでは表現することができない．このモデルのIPC は，命令レベル並列性モデルにおいて，ロード命令のレイテンシをキャッシュから得られるキャッシュ（主記憶）のレイテンシに置き換えることで得られる．

第3 に，すべてのベンチマーク・プログラムの実行を1つのワークロードとみなして，全部のプログラムから全部のプログラムに対するシミュレーション・ポイントを選出する集合的帰納手法を提案する．

帰納的手法の基本形や，SimPointでは，こうしたベンチマーク・プログラム集の個別のベンチマーク・プログラムから個別のベンチマーク・プログラムに対するシミュレーション・ポイントを選出する．つまり，異なるプログラム間では，いかなるプログラム区間同士も必ず違うフェーズであると暗黙的に仮定している． こうした手法をここでは集合的でない手法と呼ぶ．

しかし，こうした仮定は保守的である．実際には，異なるプログラムに属する2つの区間が，同じフェーズに属する可能性は十分に考えられる．サーキットの喩えで述べると，異なるサーキットであっても，区間に分けてみると，直線距離が長い，カーブが多いといった性質が似た区間があることが見つけられるだろう．

プログラム別に共通なフェーズに属する区間がある場合，こうした共有フェーズに関しては，どれか1つのプログラムにおける1つの区間だけをシミュレーション・ポイントとすることで，さらにシミュレーション時間を減らすことができる．

また，評価としてSPEC2006 でref 入力における先頭100G 命令のIPC 推定を行った結果を示す．我々の方法は，シミュレーション・ベースで，100G 命令の約0.1% をシミュレートした場合，約0.4% のIPC 推定誤差(SimPoint誤差の1/4程度)；エミュレーション・ベースの基底モデルを使用した場合，約0.9% のIPC 推定誤差を達成した．集合的帰納手法では，誤差の悪化は抑えつつ，抽出率を1/3 程度に削減することができる．