2007－ARC－174 (31)
2007／8／2

シンボル情報に基づくアプリケーション分析を目的としたシミュレータ Aice の開発

豊島隆志† 山村周史† 青木孝† 木村康則†

あらまし コンピュータシステムの研究，開発において，シミュレータの果たす役割は年々重要となっている．その用途はプロセッサの性能推定から，アプリケーションの分析まで多岐にわたり，様々な要求に応え得る柔軟な設計が必要である．一方で，シミュレーションの対象となるハードウェア，あるいはシミュレータ上で評価するベンチマークは年々肥大化する傾向にあり，必要とされる精度で対象物の全体をシミュレートすることが困難となっている．そのため，高速で精度の高いシミュレーション手法についての研究が注目されている．柔軟性や拡張性，精度と速度の両立など様々な要求に応えるため，我々はシミュレータ Aice を設計した．Aice は用途に合わせて機能を追加，変更することが容易であり，またアプリケーションのシンボル情報を扱うことにより，命令レベルの分析のみならず，関数レベルの分析が可能である．本稿では Aice を用いた評価の例として，性能情報の時系列表示を利用したアプリケーションの実行フェーズ分析と実行プロファイルを利用した関数ごとの実行コストの評価について示し，Aice の有効性について議論する．

A Symbol-aware Simulator Aice for Application Analysis

TAKASHI TOYOSHIMA†, SHUJI YAMAMURA†, TAKASHI AOKI†, and YASUNORI KIMURA†

Abstract A research and development of modern computer system requires a simulator for various purpose. Some use simulators for performance estimation of new processor architecture, and some apply them to analyze applications. To meet these demands, simulators should be designed as flexible as possible. In addition, target hardware or benchmarks are becoming larger and larger. As a result, the whole investigation using an accurate simulator becomes unrealistic. Many researchers try to establish a simulation framework to realize speed and accuracy simultaneously. We design and develop the simulator called Aice to meet these various requests namely adaptability, expansibility, speediness et al. New features are readily added into Aice, and you can modify the structure of Aice easily. Furthermore, you can achieve application analysis not only on instruction level, but also on function level as it handles symbol information included in application binaries. In this paper, we describe two evaluations with Aice. One is am execution phase analysis in time series. The other is a function cost analysis using execution profile. We also discuss the advantage of Aice noticed in these evaluations.

1. はじめに

現在コンピュータシステムの研究，開発において，シミュレータの果たす役割は年々重要となっている．特にプロセッサアーキテクチャの研究においては，マイクロアーキテクチャの性能推定，あるいはチップの消費電力推定など，あらゆる場面でシミュレータが必要となる．また，このような評価をコンピュータシステム全体に適用したいという要求もあり，大規模なシステムを評価するためのシミュレーション技術の研究が注目されている．その利用は，性能の推定だけにとどまらず，開発現場におけるゴールデンモデルとしての検証用シミュレータ，あるいはテストベンチ作成ツールとしてのシミュレータなど，様々な用途での利用が求められている．これらシミュレータに要求される性能は場面ごとに異なり，用途に合わせたシミュレータの開発が必要とされている．

本稿で発表する Aice は，様々な要求に対し最小限度の労力で適応できるよう設計，開発された，プロセッサを中心とした多目的シミュレータである．Aice は現在 SPARC V9[1] の命令セットをシミュレート可能であり，Solaris 10[2] 向けのアプリケーションを単体で実行することが可能である．オブジェクト指向に基づいてデザインされており，機能の追加，構成の変更，シミュレーション精度の切り替えなど，用途に応じた最適

† 富士通株式会社
Fujitsu Limited.

な構成の選択が可能である．また，ELF形式の実行ファイルを直接解釈することができ，動的リンクなどもAice内部で実現しているため，シンボル情報を用いたアプリケーションの解析，性能評価を行うこともできる．

本稿では，2節でシミュレータおよび性能評価について関連研究とAiceの関係を議論し，3節で本シミュレータAiceの特徴と利用方法について示す．4節では実際にAiceを利用して行った2つの評価例について説明し，Aiceを用いた際の利点について明らかにする．最後に，5節で全体についてのまとめと結論を述べる．

2. 関連研究

現在，研究目的のプロセッサシミュレータとして，最も広く利用されているのがSimpleScalar[3]である．SimpleScalarはマイクロアーキテクチャの研究目的で開発された．精度の異なる複数のシミュレータにより構成されており，着目する機能に応じて適切なシミュレータを選択することで，必要十分なアーキテクチャ評価が可能となる．例えば，詳細なシミュレーションを必要とせず，機能レベルでのプロセッサシミュレーションで十分な場合には，もっとも軽量なsim-fastを用いれば良い．一方，複雑なスーパースカラに搭載する内部機能の評価を行う場合には，sim-outorderによるアウト・オブ・オーダー実行の詳細シミュレーションを利用する．それぞれのシミュレータは別のアプリケーションとして実装されており，各々が連携をとる機能はない．そのため，評価の途中で精度を切り替えるといった評価は困難である．また，ソースコードには難解な記述が多く，可読性や拡張性で問題点を指摘する研究者もいる．あるいは，評価に用いるアプリケーションの実行ファイルを生成するためにコンパイラが必要となるが，標準のコンパイラが生成したコードは，最新のコンパイラが生成したコードと比べて大きく異なる評価結果が出るとの報告もある[4]．これはコンパイラの最適化技術が飛躍的に向上してきたことに起因する．

Aiceと同じくSPARC V9を対象としたシミュレータとしてはCmelikらによるShade[5]が挙げられる．Shadeは高速な命令セットシミュレータであり，それ自身は性能評価のための特別な機能は持たない．しかし，実行トレースを生成することで，別途トレースドリブン方式のシミュレータと協調して性能の評価を実現する．Shadeは高速なトレース生成を実現する仕組みを持つが，この仕組みはシミュレータを実行するホストマシンが，同じくSPARC V9互換のプロセッサを搭載している必要がある．その仕組みとは，ターゲットとなるアプリケーションの実行ファイルに対して，一命令ずつ実行トレースを生成するための命令列を挿入するというもので，プロセッサの挙動はホストマシンに委ねられ，正確に言えばShadeはプロセッサのシミュレーションは行わない．一方Aiceは，動作速度の点ではShadeに劣るが，どのようなシステム上でも高速にSPARC V9の実行トレースを生成することが可能である．

ところで，大規模なアプリケーションの評価を行う際，評価対象全体に対しての詳細なシミュレーションを行うことが困難になってきている．既存の手法では，シミュレーションの精度を下げるか，アプリケーションの一部を切り出して評価するなどの手法を取らざるを得ない．しかし，一般的なアプリケーションは実行フェーズに応じて性能上の特徴が大きく変化することがわかっており，シミュレーションを行う評価区間の切り出しは慎重に行う必要がある．このように，既存の手法では評価の精度を落とさずに大規模なアプリケーションの評価を行うことは非常に難しく，この様な問題を解決するためにSimFlex[6]やSimPoint[7]といった，サンプリングベースのシミュレータが研究された．

SimFlexは統計学に基づく厳密なサンプリング手法を用いたプロセッサシミュレータである．まず，アプリケーション全体をfast-forwardingと呼ばれる最低限の機能を持ったシミュレーションモードで実行し，一定間隔でプロセッサの状態をチェックポイントとして書き出す．次に，書き出したチェックポイントから一定区間について詳細なシミュレーションを行い，サンプリングした区間ごとの評価結果を統合することで，全体の評価結果とする．この際，詳細なシミュレーションはチェックポイントごとに並行して行うこともできるため，並列化による高速化も期待できる．

一方SimPointは，サンプリング区間の選定にヒューリスティックを用いることを特徴とする．SimFlexと同様fast-forwardingモードにより評価対象全体の実行を行うが，この際に評価対象の領域分割と領域のグループ化を行う．グループ化はfast-forwardingモードで取得したプロファイル情報に基づいて行われ[8][9]，性能上の特徴が似通った領域は，同じグループに分類される．従ってグループごとに1つの領域を代表とし，代表についての詳細評価を行えば，結果をグループの規模で重みづけして集計することで全体の評価結果の近似を得られる．従って，領域のグループ化の精度が，全体の評価結果の精度を左右する．

これらサンプリングベースの手法を用いた評価では，

表 1　Aice の基本機能
Table 1　Basic Specification of Aice

動作環境	Windows 系 OS，または UNIX 系 OS
命令セット	SPARC V9，または 独自仕様 RISC（SIMD 命令拡張）
OS 機能	Solaris 10 互換 API
トレース生成	4 種類のフォーマットに対応 疑似命令を用いた出力区間の指定
シンボル情報	ELF 形式実行ファイルの解釈 共有ライブラリのリンク 関数ごとの呼び出し回数，処理時間の調査 関数呼び出しの深さ調査
その他	内容別のログ生成 （標準出力，ファイル，syslog 等） キャッシュ評価 （置換アルゴリズム等，任意に構成可能） 一定区間ごとの統計出力

従来の評価に比べて 5%程度の誤差範囲に収まる．これらの手法は Aice に応用可能である．

シミュレータを用いずに実機上でアプリケーションの解析，性能評価を行うことも可能である．gprof[10] は多くの UNIX システムで標準的に利用されている実行プロファイル生成ツールである．コンパイラとして gcc[11] を想定しており，評価対象となるアプリケーションはコンパイル時にオプション「-pg」付きでコンパイルしなければならない．このオプションを用いて生成したアプリケーションは，実行後に「gmon.out」という名前のファイルを生成する．このファイルとアプリケーション実行ファイルを gprof に与えることで，関数ごとの呼び出し回数や処理時間などが得られる．ただし「-pg」付きで生成されたアプリケーションは，インライン関数展開が抑制され，各関数の導入部にプロファイル生成のための付加コードが挿入される．このため，gprof により測定した実行ファイルは，本来対象としているアプリケーションとは若干性質が異なっている．小さな関数を頻繁に呼び出すようなアプリケーションほど gprof による評価結果は誤差が大きく，アプリケーションの性質に注意する必要がある．それとは別に，サンプリングによる影響も考慮しなければならない．「-pg」付きでコンパイルされたアプリケーションは，一定時間ごとにプログラムカウンタの値をサンプリングし，プログラムカウンタ値の分布を求める．gprof はこの分布とシンボル情報を対応づけることで関数ごとの分布を求め，プログラムカウンタ値の関数ごとの存在確率をもって処理時間の比率としている．この測定は，システムや他のプロセスから外乱を受けるため，測定結果には常に数%の誤差が伴う．加えて，サンプリング周期も標準で 10 ミリ秒単位と荒く，短時間で終了するアプリケーションにはそのまま適用することはできない．また，運悪くサンプリング周期と同調して呼び出される関数が存在すると，正しい測定は行えない．同様の測定を Aice で行えば，コンパイル時に特殊なコードを挿入する必要もなく，サンプリングによる様々な問題も回避する事が可能である．

gprof のサンプリングによる実行プロファイル生成手法をシステムレベルに適用したツールとして OProfile[12] と呼ばれるツールが存在する．OProfile はシステムレベルで関数の処理時間を調べることができるほか，プロセッサの持つパフォーマンスカウンタを利用し，キャッシュヒット率をはじめとした様々な性能データを取得できる．

3.　Aice の特徴

3.1　機能追加，変更の容易な設計

Aice の基本機能について表 1 にまとめた．我々は Aice を C++言語で記述した．オブジェクト指向に基づき，機能を適切に分割し，機能単位でインターフェー

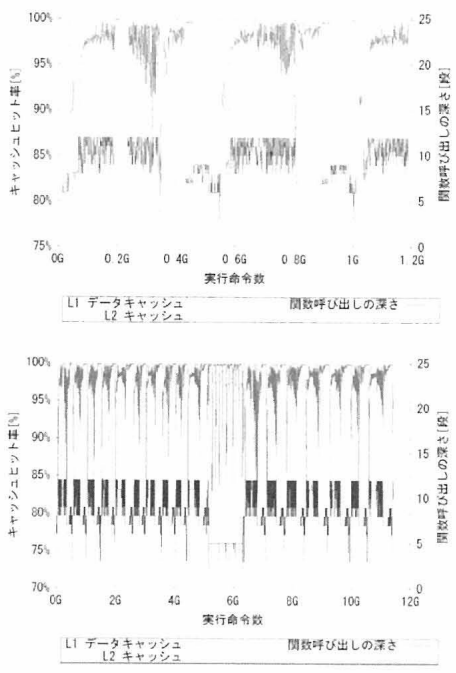

図 1　性能情報の時系列表示
Figure 1　Performance Monitoring in Time Series

スを定義している．新しい機能は派生クラスを実装することで構成を変更できる．ログ出力機能を例にとると，Aice は標準出力，ファイル出力のほか，UNIX 環境における syslog への出力，Windows 環境におけるデバッガやアプリケーション・イベント・ログへの出力などが指定できる．Aice のログ出力は，すべて基底クラス Logger を経由して行っており，新たな出力方式を追加する際には，Logger の派生クラスを実装し，ファクトリクラス経由でインスタンスを生成できるように修正するだけで良い．Aice 内部では，このような構造がいたるところに設けられており，キャッシュの置換アルゴリズムの変更から命令セットの置き換えに至るまで，必要に応じた機能の追加，変更が簡単に行える．

3.2 構造シミュレータとの協調

現段階では，Aice 自身はプロセッサの詳細なシミュレーションは行っていない．しかし，任意区間の実行トレースを出力することで，別のパイプラインシミュレータと協調してマイクロアーキテクチャの性能評価を行うことが可能である．この機能により SimFlex や SimPoint における fast-forwarding モードの代替として Aice を利用することが可能である．これにより，かつては膨大なシミュレーション時間を要するという理由で実現が困難であった，大規模アプリケーションの詳細シミュレーションが可能となった．

3.3 キャッシュ評価

Aice 単体でも，キャッシュの性能評価が行える．Aice は階層的なキャッシュ構造をシミュレート可能であり，パラメータを指定することで，任意の構成やアルゴリズムにおける性能を推定することができる．一定区間ごとの統計を取得する機能を利用すれば，経過時間に沿った性能情報の推移を調べることもできる．この機能を用いた評価については 4.1 項で具体的に述べる．

3.4 アプリケーションの直接実行と分析

Aice はアプリケーションが発行するシステムコールを，OS に代わって直接処理することで評価対象を実行する．現在 Aice が認識できるのは Solaris 10 互換のシステムコールのみであるが，トラップハンドラを別途実装することで，他の OS のシステムコールにも対応可能である．

またアプリケーションの読み込みに際して，Aice は ELF 形式の実行ファイルを直接解釈している．共有ライブラリのリンクには「`ld.so.1`」などのランタイムリンカを Aice 上で動作させる方法もある．しかし ELF ファイルに含まれるシンボル情報を利用するために，標準動作では Aice はリンク処理を直接行う．このシンボル情報は，実行中の命令が属する関数名の調査などに利用される．この機能を利用した評価については 4.2 項で説明する．

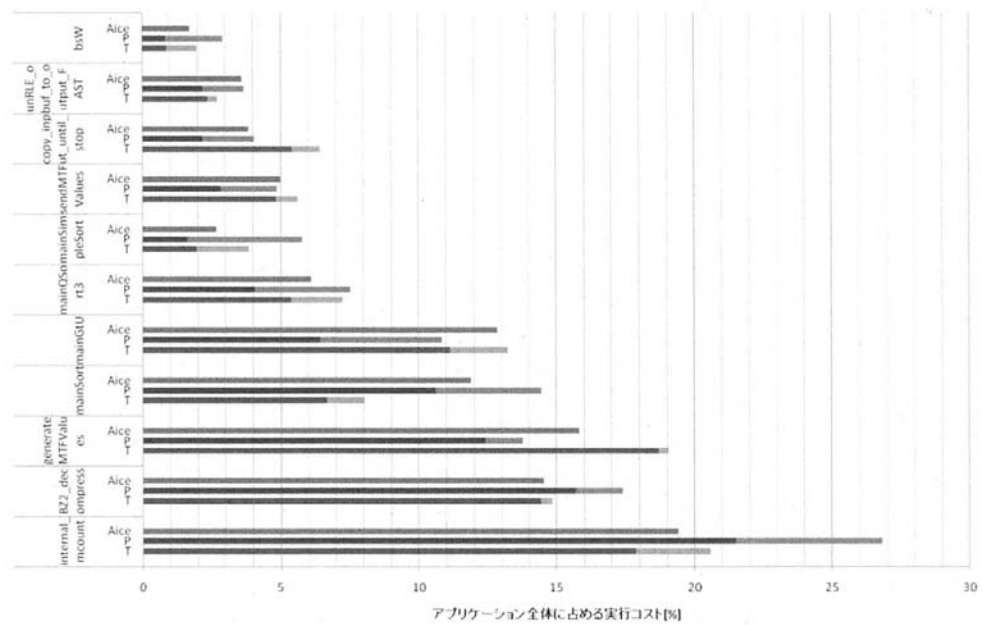

図 2 実行プロファイルの比較

Figure 2 Execution Profile Comparison between Real Machines and Aice

4. Aice を利用した評価例

4.1 性能情報の時系列表示

Aice による評価結果についてキャッシュヒット率などを時系列でグラフ化したものが図 1 である．この評価は SPEC CINT2006[13] に含まれる 401.bzip2 を対象に行った．左側縦軸がキャッシュヒット率を示し，図の上端に隣接して描かれているグラフが L1 データキャッシュ，L2 キャッシュのヒット率を示すグラフである．左端が実行開始点で，横軸に沿って右側へ進むに従い，実行開始点からの動的な実行命令数が増加する．この図からわかるとおり，401.bzip2 における性能の変化には，500 メガ命令規模の周期と 6 ギガ命令規模の周期との，2 通りの周期が存在する．右側縦軸は関数呼び出しの深さを示し，図の下端に描かれたグラフが関数呼び出しの深さを表している．キャッシュヒット率だけ見ていると，この例では 6 ギガ命令規模の周期を判別することが難しい．しかし関数呼び出しの深さと対応させることで，実行フェーズの違いが明らかになり，アプリケーションの性能情報の特徴の違いを明確に区別することが可能となる．また，性能が落ちている箇所についてシンボル情報と照合し，性能上問題となる関数を特定することもできる．

4.2 実行プロファイル

実機上で gprof により取得した実行プロファイルと，Aice により取得した実行プロファイルの比較を図 2 に示す．対象は 4.1 項と同じく 401.bzip2 である．コンパイルは gcc で行い，「-O2 -pg -fno-inline」をオプションとして与えた．グラフが示すのはアプリケーション実行中，主要な関数によって消費された時間の比率である．「Aice」の項目は Aice により取得し，「P」と「T」については，それぞれ異なるメーカーの実機上で測定した．Aice のキャッシュパラメータは

表 2　評価に用いたキャッシュの設定
Table 2　Cache Parameters for the Evaluation

L1 データキャッシュ	方式	セットアソシアティブ
	ラインサイズ	64
	ウェイ数	2
	サイズ	128KB
	レイテンシ	30 命令
L2 キャッシュ	方式	セットアソシアティブ
	ラインサイズ	256
	ウェイ数	12
	サイズ	3MB
	レイテンシ	600 命令

表 2 のように設定し，実機上での評価は，誤差があるため「P」と「T」それぞれにつき 100 回の計測を行った．グラフ中の濃い部分が 100 回の計測における最低値，薄い部分が最高値を表す．Aice ではキャッシュの遅延のみを荒く評価したデータになっているが，実行プロファイルの精度としては，十分な精度を持っているといえる．

ところで，このプロファイル結果によれば「internal_mcount」という名前の関数が最も長い処理時間を持つことがわかる．この関数はプロファイル取得のために gcc により挿入されたプロファイル生成用の関数である．このように実機を用いた実行プロファイルでは，測定用の仕組み自身がプロファイル結果に大きな影響を与えてしまうこともある．また 2 節で述べたとおり，インライン関数の展開抑制や関数導入部に挿入される計測用コードなど，様々な要因が測定結果に影響を与える．Aice により調査した計測用コードの影響を図 3 の円グラフに示す．コンパイラには gcc を利用し，コンパイルオプションとして，円グラフ A では「-O2 -pg -fno-inline」を，円グラフ B では「-O2 -fno-inline」を指定した．すなわち

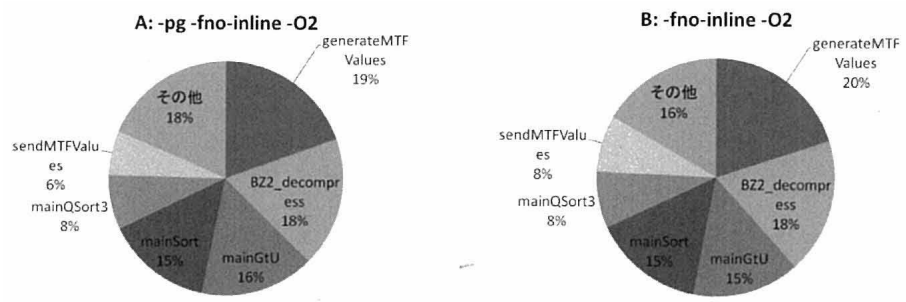

図 3　計測用コードの影響
Figure 3　Impact of Profiling

円グラフAは計測用コードが挿入された状態，円グラフBは計測用コードが挿入されていない状態のプロファイルにあたる．ともにコンパイルオプションでインライン関数展開を抑制しており，円グラフAでは計測用関数の処理時間を除いたデータを用いることで関数導入部に挿入された計測用コードの影響を調べている．円グラフAを見ると「generateMTFValues」と「sendMTFValues」が少なく，「mainGtU」が多く見積もられている．前2つの関数は呼び出し回数が25回と少なく，「mainGtU」は916万回と多い．呼び出し回数が多い関数ほど，挿入コードの影響で処理時間を長く見積もっていることが確認できる．また，この評価によって，プロファイル計測用コードの処理時間は，アプリケーション全体の21.16%にもおよぶことがわかった．

5. まとめ

本稿では，広範な利用に耐えられるよう設計，開発された，汎用プロセッサシミュレータAiceについて説明した．Aiceは目的に応じた機能の追加，変更ができるようオブジェクト指向に基づいて設計されており，最小限度の労力で必要に応じた修正が行える．またAiceは，ELF形式の実行ファイルを直接解釈する機能を持ち，共有ライブラリのリンクを直接行うことで，シンボル情報を用いたアプリケーションの評価，分析を行うことができる．

4節ではAiceの機能を用いた評価について2種類の例を示し，Aiceの有効性について議論した．1つ目の例は性能情報の時系列表示であり，関数呼び出しの深さやシンボル情報と関連付けて，アプリケーションの実行状態の推移について解析する例である．2つ目の例はgprofに代わる実行プロファイル取得方法であり，この例ではAiceを用いることで，計測対象に影響を与えることなく，実用上問題ない精度の実行プロファイルを生成できることを確認した．この評価によれば，gprofにより実行プロファイルを取得するために挿入されたコードのオーバーヘッドは，401.bzip2において全体の21.16%にもおよび，測定結果に大きく影響を与えていることもわかった．

今後は，Aice単体でのより詳細な評価環境の開発，パイプラインシミュレータと組み合わせた高速な詳細シミュレーション環境の評価，より進んだアプリケーション実行フェーズ分析の研究などを行う．また，より利用しやすい形でのアプリケーション分析情報の表示方法についても検討を進めたい．

参考文献

[1] SPARC International, Inc., "The SPARC Architecture Manual, Version 9", *http://www.sparc.org/*

[2] Sun Microsystems, Inc., "Solaris 10 Operating System", *http://www.sun.com/software/solaris/*

[3] SimpleScalar LLC, "SimpleScalar version 3.0d", *http://www.simplescalar.com/*

[4] 豊島隆志，入江英嗣，五島正裕，坂井修一，"レジスタ間接分岐ターゲット・フォワーディング"，*先進的計算基盤システムシンポジウム SACSIS 2006*, pages 325-332, 2006.

[5] Bob Cmelik, and David Keppel, "Shade: A Fast Instruction-Set Simulator for Execution Profiling", In *Proceedings of the 1994 ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems*, pages 128-137, 1994.

[6] Thomas F. Wenisch, Roland E. Wunderlich, Mike Ferdman, Anastassia Ailamaki, Babak Falsafi, and James C. Hoe, "SimFlex: Statistical Sampling of Computer Architecture Simulation", In *IEEE Micro special issue on Computer Architecture Simulation*, pages 18-31, July-August 2006.

[7] Michael Van, Biesbrouck, Brad Calder, and Lieven Eeckhout, "Efficient Sampling Startup for SimPoint", In *IEEE Micro special issue on Computer Architecture Simulation*, pages 32-42, July-August 2006.

[8] Timothy Sherwood, Erez Perelman, Greg Hamerly, and Brad Calder, "Automatically Characterizing Large Scale Program Behavior", In *Proceedings of 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 02)*, ACM Press, pages 45-47, 2002.

[9] Joshua J. Yi, Sreekumar V. Kodakara, Resit Sendag, David J. Lilja and Douglas M. Hawkins, "Characterizing and Comparing Prevailing Simulation Technoques", In *Proceedings of 11th International Symposium on High-Performance Computer Architecture (HPCA 05)*, IEEE CS Press, pages 266-277, 2005.

[10] Susan L. Graham, Peter B. Kessler, and Marshall K.Mckusick, "gprof: A Call Graph Execution Profiler", In *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction*, SOGPLAN Notices, Vol. 17, No 6, pages 120-126, 1982.

[11] Free Software Foundation (FSF), "GCC, the GNU Compiler Collection", *http://gcc.gnu.org/*

[12] John Levon, Philippe Elie, et al., "OProfile", *http://oprofile.sourceforge.net/*

[13] Standard Performance Evaluation Corporation, "SPEC CINT2006", *http://www.spec.org/cpu2006/CINT2006/*