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

2006－ARC－169 (23)
2006／8／2

命令シミュレーション手法を用いた性能シミュレータ

近江谷 康人[1,2] 天野 英晴[2]

[1]三菱電機株式会社情報技術総合研究所, [2]慶應義塾大学理工学部

高性能なマイクロプロセッサを搭載した組込み機器開発では，パイプラインストール，メモリウォール問題，ソフトウェアの肥大化などにより，製品のシステム性能の見積りと達成が困難になって来ており，ソフトウェア改善に適応できる性能解析ツールが求められている．本稿では，命令シミュレーション手法を用いることにより，プロセッサ，キャッシュメモリ，プログラム動作を解析するツールESPRIT/simを開発したので紹介する．ESPRIT/simは，多種ISAに対応した異種マルチプロセッサ動作，プログラム情報との連携，関数や分岐の他に“データプロファイラ”と呼ぶメモリ変数の解析機能を搭載している．キャッシュメモリや主記憶などの構成変更，動的バイナリ変換からパイプラインシミュレーションに到るシミュレーションエンジンの切り替え機能による高速性と精度の両立，HTML表示による簡易なユーザインターフェイスにより，対話的な実行環境を実現している．

ESPRIT/sim: A Performance Analyzer Using Machine Instruction Level Simulator

Yasuhito OHMIYA[1,2] Hideharu AMANO[2]

[1]Information Technology R&D Center, Mitsubishi Electric Corporation [2]Department of Science and Technology, Keio University

Nowadays, developing of embedded-systems is becoming more difficult job, in the area of forecasting and achieving system performance, by pipeline-stall, memory-wall issues lying on enlarged-software and high-performance microprocessors. To solve this issue especially for software improvement, this paper introduces a simulator tool “ESPRIT/sim”, which interprets binary instruction codes, mimic and analyzes processor, cache memory and program behavior. Its features are heterogeneous multi processor system simulator of various ISAs, “Data profiler” that records and summarizes history of each memory variable, as well as functionality of program information annotation. ESPRIT/sim also realizes an interactive environment by re-defining cache and memory specifications, employing multiple types of simulation-engines high-speed dynamic binary translator through clockwise pipeline simulator, and using HTML-based visualization.

1. はじめに

マイクロプロセッサの性能向上は，周波数の向上，多段パイプライン，スーパスカラ，オンチップキャッシュメモリなどによるところが大きい．その反面，プロセッサを組み込んだシステム全体の性能を事前に見積ることが困難となり，開発過程における性能改善に開発コストが掛かるようになってきた．特に組込み機器では，製品コストや発熱などの制約から，主記憶周りの性能が抑えられメモリウォール問題による性能低下が発生しやすいため，ハードウェアの特性に合うようにソフトウェアの改善も求められる．

ソフトウェアのチューニングには一般にプロファイラが使用されているが，最適化コードには適用できないため正確性に欠け期待したほどの効果が出ない，原因解析に必用な情報を提供できない，短周期のタスク切り替え時のキャッシュミス問題解析に使用できない，といった難点がある．一方，ハードウェアを含むシステムシミュレータは処理が遅いため大規模ソフトウェアには対応できない．

本稿ではそのような開発環境の課題に対し，市販のマイクロプロセッサを搭載した組込み機器開発を対象として，プロセッサの選択，オフチップメモリの適正化，ソフトウェアからメモリへのアクセス方法やソフトウェアの構造の改善，コンパイラオプションの精査を可能にするシミュレータESPRIT/simを紹介する．

本稿で提案するツールは，最適化された機械語コードを1命令ずつシミュレーションし，性能の支配項となるキャッシュメモリと外部メモリの挙動の統計情報を残すものである．このようにシミュレーションベースで，解析する手法としては，トレースドリブン方式のものが多いが，ESPRIT/simは，命令シミュレータを核に各種リソースのシミュレーション機能を搭載しオンザフライで解析している．

筆者らは従来，CPUや組込み機器開発向けに個別にシミュレータや解析ツールをC言語などで用意していたが，それらの機能を統合して汎用化したものがESPRIT/simである．ESPRIT/simはC++にて実装され，多種の命令セットアーキテクチャ(ISA)に対応しかつそれらの混在を可能にした．また，システムによっては数十ギガ命令までを対象とするものもあるため，シミュレーションエンジンを動的バイナリ変換使用の高速のものからクロックサイクルレベルのシミュレーションまで，途中で切り替えながら実行し統計を採ることが可能である．シミュレーション対象のハードウェアは，C++で作成されたクラスライブラリの組み合わせや派生クラスによりカスタマイズできる．最適化後のソフトウェアを分析対象とするため，グローバル変数情報を用いプログラム階層のトラッキングを行い解析する．ユーザインターフェイスとしては，コマンドインタプリタを中心に，各種の実行・デバッグ支援機能と表やグラフ化機能を充実している．キャッシュやメモリの仕様は実行時のコマンドで再構成を行い性能の比較が容易に出来る．

本稿の構成を示す．2.では，関連研究について述べ，3.では性能シミュレータの開発ターゲットについて述べる．4.ではESPRIT/simの実装について述べ，5.ではその結果についての評価を示す．6.でまとめる．

2. 関連研究

コンピュータシステムの性能分析の手法として，文献[1]に紹介されているようにトレースドリブンのシミュレータが多数研究開発されてきた．ESPRIT/simの高速実行を狙った動的バイナリ変換[6]はShade[2]，フルシステムシミュレーションはSimOS[3]と類似であるが，ESPRITはトレースを使わずにオンザフライで性能シミュレーションを行なう．SimpleScalar[4]をベースとしたシミュレータの研究[8][9]も

多いが，ESPRIT/simはシミュレータの汎用・共通化という点でSimpleScalarと似ている．SimpleScalarはsim-fast, sim-cache, sim-outorderと目的別にカスタマイズしており，モデルの可読性とマルチプロセッサへの拡張性が課題である．またコンパイラとライブラリをカスタマイズしているため，一般のソフトウェアユーザが使用するISAやバイナリコードには対応できない．C++のクラスライブラリを用いたマルチプロセッサのモデル化の利点はISIS[7]と同様であるが実装の違いで性能レンジが異なっている．

これらの研究用シミュレータはコンピュータアーキテクチャの専門家以外の使用に対しては，性能，拡張性，使い易さのバランスに課題があり，更に性能の劣化要因であるソフトウェア部位の特定や改善後の性能予測を行なうツールとしての機能が不足している．商用のシミュレータでは使い易さが改善されているが，市場の大きい携帯電話器で使用されている特定アーキテクチャに限定されているなど，一般の組込み機器開発に便利なツールは殆ど無い．

本稿で紹介しているESPRIT/simは，市販のマイクロプロセッサを使用した機器の性能問題を解決できるツールとして，今後主流となる異種マルチプロセッサもその対象の範囲としている．

3. 性能シミュレータの開発ターゲットと課題

性能シミュレータESPRIT/simの開発にあたってそのターゲットにした仕様を述べる．また，併せて課題を紹介し，解決策は4で後述する．

3.1. 性能シミュレータの位置付け

性能を分析してボトルネックを改善し設計仕様にフィードバックする目的としてのツールでは，設計着手前にアーキテクチャレベルで事前評価を狙うものと，ベースアーキテクチャへの改良評価を行なうものがある．本稿で紹介するESPRIT/simは後者に相当する．

ESPRIT/simは，バイナリコードを入力として，シミュレータに記述されたモデルをベースに，メモリやキャッシュ構成をパラメータとし，ソフトウェアを含むシステム性能について分析可能なシミュレータである．

3.2. 性能シミュレータの狙い

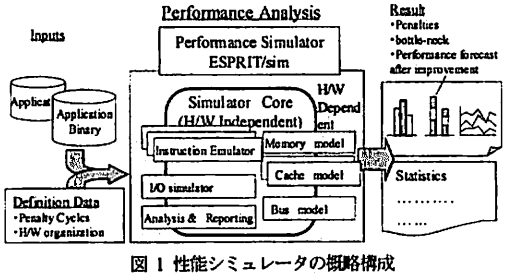

図1 性能シミュレータの概略構成

ESPRIT/simは次の特長を持つ．

1）思考結果をインタラクティブに試せる高速性能．
2）個別の要因が絡み合った結果である処理時間を，直感的かつ理解容易にする単純化された遅延モデル．
3）構成の変更やモデルの拡張を容易にする自在性．
4）今後，肥大化が想定されるデータ処理の改良に対応した，メモリ変数のプロファイル(データプロファイラ)機能の搭載

3.3. 精度と速度

性能分析の対象時間範囲は絞られるものの，シミュレーション対象システムによっては，数十M～数十G命令の実行が必要なものがある．実際に処理速度が問題になるのは，

1） 分析前の準備段階としての試行錯誤作業
2） 分析対象の状態までの立ち上げ時間

である．シミュレータに対する精度と実行処理速度は原理的に相反する．そこで，図2のように複数のシミュレーション方式による「高速化エンジン」を併用する．特に途中で切り替えを可能にし，実質的な高速性と精度を実現する．

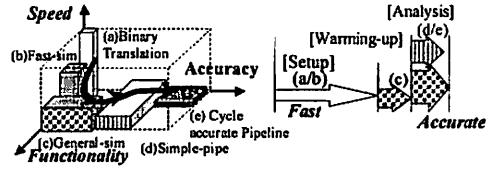

図2 シミュレーション方式の併用

3.4. ISA，ホスト

対象とするプロセッサのISAは追加可能であり，それらを組み合わせたマルチプロセッサ構成も可能とする．メモリ，キャッシュメモリ，統計情報の計数と表示，コマンド入力などは共通仕様とする．従来から，シミュレータ実装にはC言語が多用されているが，変数や関数のスコープの制約があり異種・マルチプロセッサへの拡張の障害となりC++への移行をする．現時点では，組み込用途に限定し，実装が容易なRISC，アドレス空間は32ビットとし実装を軽くする．

ホストは，PC，EWSを対象とし，移植とインストール容易な構造にする．

3.5. 統計情報

統計情報は実行時に選択でき，プログラム変更により機能追加容易な構造を採る．統計用のカウンタは32ビットを越えり値の計数を考慮する．アクセス回数やミス回数などのカウンタは時系列によりグラフ化に対応する．

3.6. ソフトウェア統計情報

関数の情報としては親関数と子関数のペアごとに，呼出し回数，命令数，概略サイクル数の統計を採る．また関数ごとに論理メモリへのアクセス回数，各キャッシュミス回数を計る．分岐では，分岐発生の関数，分岐予測ミス回数，分岐バッファミス回数の計数を行なう．これらの実現のため，各命令が実行している関数を特定する必要があるが，コンパイラによる最適化後やアセンブリ記述されたライブラリのコードでは次のようなケースが課題となる．

- 子関数Bが孫関数Cを呼んだ後に親Aにリタンするソースの機械翻訳において，BからCにジャンプしCからAの直接リタンするようにコード生成される場合．
- 多階層の関数からまとめてリタンする場合
- マルチタスクのコンテキスト切り替えの場合

3.7. データプロファイラ

gprof[5]などに代表されるプロファイラは関数の呼出し回数や処理時間の統計を採っているが，変数アクセスに関する統計はない．プロファイラに変数アクセス統計の機能追加をすると，処理時間が3桁ほど増加すると予想される．それは関数内部でアクセスされるデータが関数の数より1～2桁多く，また統計として使えるように索引付けする処理が1桁程度余分に掛かるためである．特にポインタ間接によるメモリアクセスがあるため，コンパイル時にプロファイル用コードの埋め込みが難しいと考えられる．一方，命令レベルのシミュレータでは実行時にデータアクセスをモニタし統計を採るのは原理的に容易である．

ESPRIT/sim ではプログラム変数の挙動統計を採る「データプロファイラ」の実装を試行する．今回の実装ではレジスタの計数は行なわず，性能上のボトルネックになるメモリ変数についてのみ行なう．

データに関しての課題は，何をどのような粒度で計数し，何と関連付けどのように表示すれば，直感的で改善に結び付くかということであり，そこに注目した実装を試行する．性能チューニングは別途行なう．

3.8. OS モードと APP モード

アプリケーションを対象として論理アドレスレベルで行なうシミュレーションと，OS 込みのフルシステムシミュレーションの両方に対応する．前者に対しては論理アドレスと物理メモリのマッピング指定と擬似システムコールを用意する．フルシステムシミュレーションでは割り込み，アドレス変換，制御命令を用い，I/O はシリアルコンソール，タイマーなど個別にカスタマイズする．

3.9. プロセッサモデルの信頼性

同一 ISA アーキテクチャに限定したトレースドリブン方式ではシミュレーション部分が少ないのに対し，クロスアーキテクチャシミュレーションや命令レベルシミュレーションでは，命令機能シミュレーションの正確度が高くないと誤動作し実用レベルに到らないことが最大の課題である．エミュレータ製品開発は通常，自社製 ISA を対象とし充分な検証コストを掛けられかつ試験データも豊富である一方，本システムではマニュアルをベースに市販プロセッサのモデル開発を行い，試験データも自前で用意する必要があり，費用効果が課題となる．

4. 実装

本章では ESPRIT/sim の実装結果と，課題解決について述べる．

4.1. ESPRIT/sim の構成

ESPRIT/sim の構成は図 3に示すように，複数の ISA を包含し，キャッシュメモリ，メモリ，システムコールシミュレータ，統計機能，コマンドインターフェイスなどは共通化されている．

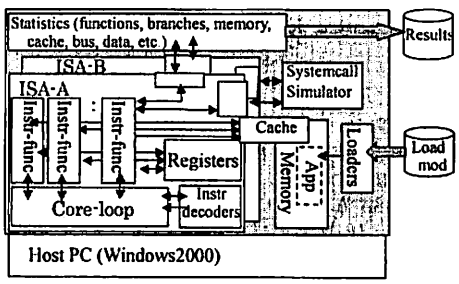

図 3 ESPRIT/sim の構成

4.2. C++実装

ESPRIT/sim では，当初想定していた C 言語実装から C++ へ実装を変更した．その利点を下記に示す．

1) namespace やクラスを使用することにより ISA 間で同一の変数名や関数名を使用できる．
2) namespace の活用により同一 ISA に対し，高速処理向けと簡易パイプライン統計処理向けなどに命令実行関数を複数用意する時に，ソースコードを 1 本化できる．#define ReadReg(n)のようなレジスタアクセスを行なうマクロ記述を複数用意し，共通のソースコード部分を #include することにより可読性とデバッグ容易性を損なわずに複数のコードへの展開が出来る．
3) クラスの使用により，複数の ISA やプロセッサの多種化時のモデル管理による煩雑さから解放され，派生モデルの保守も容易になる．
4) コンストラクタによる初期化機能の利用により I/O など周辺モデルの実装がより容易になる．

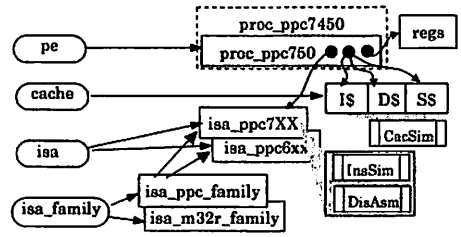

図 4 クラスを用いたプロセッサの実装例

図 4に示すように各プロセッサは基本クラス pe(プロセッサエレメント)の派生形としてリソースが定義され，プロセッサ依存部分は仮想関数として実装している．関数の実体は主に，isa と isa_family から派生した isa_＊にある．これは，複数プロセッサ間のモデル記述の重複を避け保守や改良を容易にするためである．

4.3. 高速化エンジン

ESPRIT/sim ではシミュレーションの精度と高速性のトレードオフ実現のため，複数のシミュレーションエンジンを搭載し，途中で切り替えることにより，ユーザーの速度に対する不満を解消している．シミュレーションエンジンは，(a)動的バイナリ変換，(b)高速命令インタプリタ，(c)解析用シミュレータ，(d)擬似パイプラインシミュレータ，(e)クロックレベルパイプラインシミュレータに分類される．

(a)と(b)は，模擬対象のメモリ空間に対応したリニアなメモリ空間を用い，統計機能やデバッグ機能も限定している．(c)では 4GB の全空間に対応し，関数のトラッキング，表と簡易式による概略クロック数評価とオプションにより指定された統計を採取する．(e)はスーパスカラ動作などを忠実にシミュレーションするクロックドリブンのシミュレータであり実行ステージのみ命令関数を呼び出す．(d)は命令シミュレータに，パイプライン干渉を簡易実装したもので，命令デコード表に基づいたペナルティ加算，データ遅延やリードアフターライト検出などの機能がある．命令のプリフェッチ挙動は測定できないが，実装コストが安い．

(e)のクロックレベルのモデルでは正確な処理時間が計算できるが，遅延要因の解明が困難となる．そこで，通常の解析に使用される(c)と(d)では多項近似式により遅延のモデル化を行い，処理時間(T)を表現する．Cx は実行回数，Px はペナルティ時間で，i は命令種，c はキャッシュアクセス，m は物理メモリアクセス，b はバスアクセスを示す．

$$T = \Sigma Ci \times Pi + \Sigma Cc \times Pc + \Sigma Cm \times Pm + \Sigma Cb \times Pb \ldots$$

マルチプロセッサ構成のシミュレーションは，時分割で順番に個々のプロセッサのシミュレーションを切り替えており，切り替え間隔（命令数）も指定できる．ホスト環境に依存せずに実装できデバッグが容易なことからシングルスレッドで動作している．

シミュレーション方式を(e)から他に切り替えるときは，逐次化により新規命令の命令フェッチを抑止し，パイプラインが空になるまでクロックを進めることにより連続動作を保証している．

シミュレーションエンジンのうち（b）（c）以外はオプ

ションとしている．特に(e)は実装とデバッグに数ヶ月以上掛かり，特に精度の検証と保守に費用が掛かる．

【動的バイナリ変換】

動的バイナリ変換方式は，常に高速インタプリタと連携動作し，分岐アドレスごとの頻度を計数し指定閾値以上ならば一連の命令列をホスト命令に変換・生成し，コードキャッシュと呼ぶメモリに格納して，以後はそのコードを利用するものである[6]．また，未変換部分，低位頻度の命令とシステムコールは高速インタプリタを利用している．

動的バイナリ変換(a)では，高速化のレベルを次のように分類し，開発コストと性能のトレードオフを実現している．高速インタプリタコードを呼び出すコードのみを生成する(a-1)，複数命令に渡ってコード生成するが命令間に渡った最適化をしない(a-2)，プログラムカウンタなどはまとめて更新する(a-3)，条件分岐命令後のインライン命令列の変換を継続(a-4)，コードキャッシュ構成，ハッシュ方法，個々の命令の最適化まで行なう(a-5)．

命令のバイナリ生成を行なうトランスレータは，次の3階層構造により，可読性を損なわずにコードの共通化と流用性を高めている．(1)インタプリタのソースコードを改造し，レジスタやメモリへのアクセス，各演算などをホスト間共通のマクロ関数の並びに記述した関数．(2)そのマクロ関数をホスト依存の関数呼出しに変換するマクロコード．(3)バイナリ生成するISA共通かつホスト依存の関数．本方式は，表駆動方式の実装に比べ変換動作のデバッガ利用に利点がある．コード生成時に呼び出されるトランスレータのメイン部分も，ISA間のコード流用が容易である．バイナリ変換のデバッグコストは，インタプリタより1~2桁余分に掛かるため，ホストをPC(x86)に限定し，ISA，最適化レベルは開発費用とのトレードオフで決めることができる．

4.4. 統計情報

統計情報は，命令種類，ソフトウェア情報(関数，分岐)，データプロファイラ，論理メモリアクセス，物理メモリアクセス，外部バス，TLB，キャッシュメモリ（分岐バッファを含む），各種ペナルティサイクル数など13グループあり，約120項目に及ぶ．各種の性能遅延要因は，頻度の他，クロックサイクル数またはCPI(Clock cycles Per Instruction)として全体に占める割合を表現する．また，パラメータの変更後との比較グラフ生成機能より改善効果が直感的にわかる．

4.5. ソフトウェア連携

ESPRIT/simでは各実行命令が所属する関数を特定する課題に対し，分岐時に“関数の範囲を越えた分岐”の検出と，“リタンアドレスの監視”をする機能を設けて解決している．マルチタスク向けには複数のスタックスロットを設け，図5のように処理している．関数のラベルとサイズはELF (Executable and Linkable Format)ファイルより取得しているが，外部指定もできる．

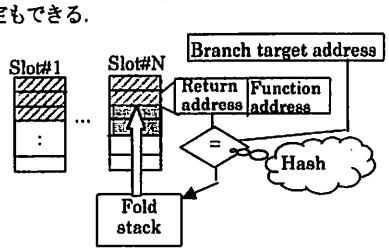

図5 関数のトラッキング機構

4.6. データプロファイリング

【計数単位の粒度】データのプロファイル計数単位の粒度に関する課題について述べる．4バイト単位に全データへのアクセスを監視するとカウンタの数が関数より2桁は多くなる．また量が膨大なため意味を理解する上で困難となる．ESPRIT/simでは，構造体や配列を1つの変数としてまとめて計数を行いカウンタの数を大幅に減らした．一方，大きな配列や構造体の無駄な使い方の調査には各要素のアクセスを個別に計数する必要がある．そこで標準機能として，要素のアクセスされた上限と下限を関数ごとに計数し，更にオプションとして指定された変数のみ要素ごとのアクセスを計数するようにした．

【変数情報】ロードモジュールの形式によっては変数のサイズが得られず，またELFでも配列の要素数までは情報が無い．そこで，隣接アドレスのラベルからの自動推定と，それらの情報を追加指定する機能をESPRIT/simに設け，デバッグオプション付きでコンパイルしたモジュールより外部抽出または人手で指定により情報をアノテートする．

【データの意味付け】データの持つ意味は，ロードモジュールのTEXT領域ならば定数またはプログラムに大別できるが，コンパイラが生成しDATA領域に散在して置かれるポインタの意味を人間が理解するのが難しい．そこで，ESPRIT/simでは，変数ごとにデータ値の種類を計数し，表示時にその値が変数や関数のアドレスと一致するかどうかを検査して推定情報として表示している．

【高速化】データプロファイラでは，図6に示す変数のラベル探索では時間が掛かるため，ハッシュポインタを張り2回目以降のアクセスを高速化している．また，データ量が多い場合にはハッシュ用の多数のポインタリンクによりホストのキャッシュメモリを汚染し性能が低下するため，キャッシュメモリにフィットする連想バッファを併用する．

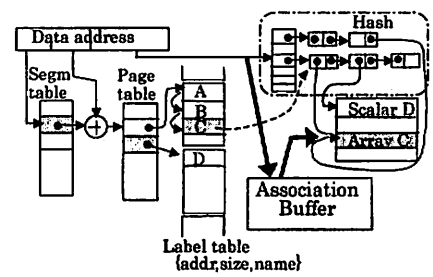

図6 データプロファイラの高速化

4.7. ESPRIT/sim へのISA実装状況

ESPRIT/simに実装済みのISAは，PowerPC[10]，M32R[11]，SH4[12]，MIPS(32/64ビット)，ARM(一部)である．制御命令の実装はISA依存であり，クロックレベルのパイプラインシミュレータの移植実績はPowerPC750のみである．

4.8. モデルのカスタマイズ

ISAの種類の新設，新規プロセッサの追加，プロセッサの組み合わせ変更にはソースコード変更が必須である．C++化によるソースコード統一のため既存モデルを雛型とした追加は容易であり，派生モデルによる実装は追加コード量が少ない．なお，メモリ空間と物理メモリの対応の変更，メモリ遅延の変更，キャッシュ構成や属性(ライトバック・ライトスルーなど)はコマンドレベルで変更できカスタマイズは不要である．

4.9. ユーザインターフェイス

シミュレータの起動，統計情報の指定など全てコマンドインタプリタにより行なっている．アドレスの指定などはラベルを使用した計算式にて記述でき，バイナリへのパッ

チは，内蔵の簡易アセンブラを用いてコマンドスクリプトでアセンブリ言語記述する．ブレークポイント，トレース，ダンプなどシミュレーション時の実行支援をしている．

統計情報は，リスト出力の他に，スプレッドシート用にCSV 出力をしており時系列グラフの作成にも用いられる．表示の判りやすさと操作性向上のため，表と棒グラフはHTML のみで出力，他のグラフはビットマップ出力してブラウザ経由で実行状況をモニタできる．

4.10. 実行環境

ESPRIT/sim の実装は，コンパイラやライブラリへの依存度が低く，ホストやOS との独立性が高い．Windows2000(x86), を保守対象としているが，Linux(x86)，Solaris(sparc), BSD 系の OSX(PowerPC)への移植実績がある．

4.11. 検証方式

命令インタプリタとバイナリ変換部の検証を目的とし，汎用の検証データをC 言語で記述し適用している．この方式により試験データのデバッグは任意のホスト上で短期間に正確に行なえる．各種 ISA でのカバレッジを上げるために，定数の範囲は 2, 4, 5, 8, 12, 16, 24, 32 ビットなど一般的な命令フィールドを想定している．キャリーなどの倍長演算はANSI -C99 の long long によりライブラリ込みで試験している．コンパイラがサポートしていない命令や制御命令は別途個別に試験する．マクロ記述を用い表 1に詳細を示すように 5.7K 行で 1600 の項目を試験できている．

表 1 汎用の検証データ

項目	本	行	項目	本	行
算術演算	278	592	論理演算	42	191
メモリ	87	657	定数・シフト	152	491
乗除算	251	758	比較・分岐	473	2032
浮動小数	270	533	C 言語固有	計画中	-
倍長演算	19	59	(合計)	1572	5767

4.12. コード量

表 2に ISA 別と機能別にソース行数を示す．ISA 1 つに着目すると67～83%が共通化されていることになる．PowerPC ではパイプライン動作記述の割合が高い．動的バイナリ変換と命令定義に含まれるデコード表(1 命令 1 行)を除くとISA 他 ISA を参考に記述は容易である．

表 2 シミュレータのコード量(ライン数)

ISA	ソース(KL)	
ISA 共通	28.6	38%
PowerPC	14.3	19%
M32R	11.1	15%
ARM	9.5	13%
MIPS	6.4	9%
SH4	6.1	8%
計 75.9		

分類	ソース(KL)	
命令定義+実行	26.0	34%
実行支援	11.5	15%
ISA 固有データ+実行	11.0	15%
統計,各種表示	8.6	11%
共通処理	8.3	11%
パイプライン*	4.7	6%
動的バイナリ変換*	2.6	3%
I/O*，遅延計算,他	3.0	4%

* 実装の有無は ISA 依存

5. 評価

5.1. 性能

高速命令インタプリタと統計情報ごとのシミュレータの性能例を表 3に示す．本例では ISA は PowerPC，ホストは Pentium-M 800MHz(左 3 列)と Xeon1.7GHz(右 2 列)，アプリは SpecCINT[13]の 099.go の計測結果である．高速インタプリタはC言語実装(b-)に比べコア動作版(b0)で30%ほど低下している．これは，CPU のリソースをクラスで定義したため ISA レジスタ 1 個へのアクセスに必要なホストのロード命令数がC言語の2倍に当たる4回に増えた事などによる．ブレークポイント，ステップ実行，ダンプ機能を使用できる実用版では(b-)の 64%に留まる．シミュレータ(c)では，関数のトラッキングやブレークポイント，各種の統計オプション判定のオーバヘッドなどがあり，オプション次第で，高速命令インタプリタ(b)の 1~3 割の性能になる．

表 3 シミュレーションレベルと性能の関係

	動作条件	性能	MIPS	SD	性能	MIPS
b-	高速インタプリタ(C 言語)	100%	12.3	38	100%	18.8
b0	高速インタプリタ(C++)コア動作	62%	7.7	61	75%	14.1
b	高速インタプリタ(実用版)	62%	7.6	62	69%	12.9
c	シミュレータ(関数トラッキング付き)	20%	2.47	190	21%	3.9
	シミュレータ(統計個別オフ)	12%	1.49	316	12%	2.2
	同(キャッシュ，分岐予測)	12%	1.49	316	12%	2.3
	同(プログラム統計)	10%	1.23	381	8%	1.5
	同(全統計，除くデータプロファイル)	6.4%	0.79	594	6%	1.2
	同(データプロファイラ)	6.5%	0.80	590	2%	0.5
e	3 ウェイスーパスカラ	1.2%	0.14	3263	2%	0.3

SimpleScalar (SS)との性能の直接比較は，対象 ISA が異なることとホストの準備の課題となり実現できていない．参考値として，ESPRIT/sim は PowerPC, SS は Alpha での 099.go の sim-fast の実行結果[9]を換算し 2.4 倍(b-)と 1.6(b)倍の性能となる．同じく SS は pisa[15]での Splash2[14]の Radix 実行性能(Pentium4 2.5GHz)の換算では 2.4 倍(b-)から 1.4 倍(b)となる．SS のキャッシュ分析 sim-cache は，ESPRIT/sim では分岐バッファや分岐予測ミスもまでも対象としているためか，SS の 95%の性能に留まっている．

次にシミュレーションエンジンに動的バイナリ変換を用いた時の性能の例を示す．本例では ISA は M32R，ホストは Pentium-M 800MHz，アプリは Dhrystone と SpecCINT の 099.go の計測結果である．図中の記号は4.3で示した記号に対応する．なお，レジスタのホストレジスタへの割り付け，命令間に渡る最適化，分岐のターゲットとインラインの入れ替えは行なっていない．なお，C 言語から C++へ実装変更による性能低下はバイナリ変換された部分にはないが高速インタプリタ部の影響により約 13%ある．

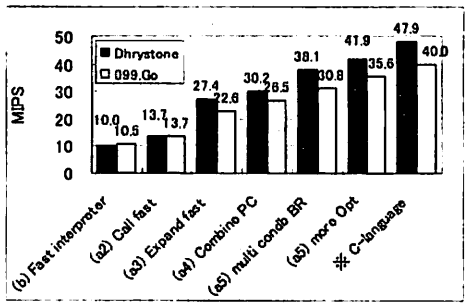

図 7 動的バイナリ変換エンジンの改良効果

5.2. マルチプロセッサシミュレーション性能

マルチプロセッサ動作時のオーバヘッドとして，Splash2 のradix ソートでの評価結果を図 8に示す．ISA はPowerPC，ホストは PentiumIII-M 800MHz である．左のグラフは百万件のキーのソートに対し CPU 数をパラメータとしたもので各シミュレーションの切り替え間隔は 100 命令，1 CPU の性能は 8.5MIPS である．右のグラフはキー10 万件に対し各シミュレーションの切り替え間隔の命令数を変化させたものであり，切り替えオーバヘッドは小さいと言える．

5.3. ユーザインターフェイス

HTML 化した統計情報をブラウザ経由で定期表示するこ

とにより時系列の変化も直感的に判る．速度面ではスレッドシートのグラフ表示がシミュレーションより時間が掛かるのに対し，HTML 出力とブラウザ表示は高速でストレスを感じさせない．また，HTML のハイパーリンク機能により必要な情報同士の相互参照が可能であり，パラメータや ISA を変更したインタラクティブな試行に臨場感を与える．

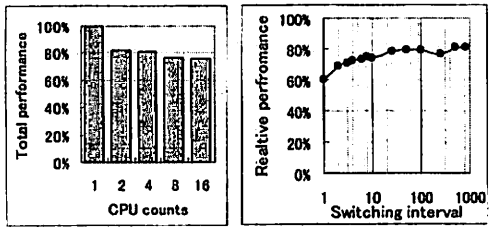

図 8 Multi-CPU simulation overhead

5.4. 障害発生

命令機能の障害は，4.1l で述べた手法により効率的に排除できている．新規 ISA 追加時には命令仕様の誤解を含め有効である．動的バイナリ変換など命令シミュレーションの高速化時と機能劣化時の原因究明には更に効果がある．この手法で漏れるものには，アドレスの依存性のあるものとレジスタ番号に依存する障害，並びにコンパイラが生成しない命令と元々対象外の制御命令があり，ISA により変動はあるが命令セットの約 50%をカバーし，C 言語記述による評価プログラムの95%ほどの命令はカバーできている．

5.5. ESPRIT/sim の適用事例

【事例１】組込み機器の現行製品の分析を行いメモリとキャッシュメモリ属性のオーバヘッドからその改善効果の算定に適用した．現行機と前機種は同系列の CPU を使用しているが，前機種では CPU 内蔵キャッシュが小容量かつダイレクトマップのため外部 SRAM とライトスルーを使用していた．現行機では同キャッシュの２ウェイ化によりヒット率が上がり SDRAM とライトバックの組み合わせが最適と確認した．また次期製品向けに別 ISA での評価も実施した．

【事例２】C 言語で実装した命令エミュレータの性能分析に適用した．ボトルネックを統計的に分析し効果算定しながら改善を試行した．図 9に効果の変遷(左→右)を示す．棒グラフは Dhrystone と 099.go の CPI，折れ線は性能向上値．

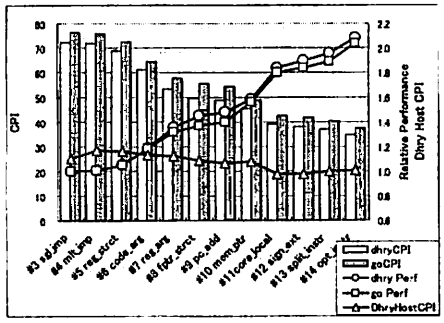

図 9エミュレータ改善に適用した事例

この例では，コンパイラが生成したポインタ変数の読み出し回数とその意味の認識，アクセスされた変数の関数ごとの統計により改善効果の判定と改善漏れの検出，分岐ごとの分岐予測ミス率によるデコードとコアループ構成の変更を行なった．これにより，試行錯誤ではなく体系的に方式改良を実施できた．

6. むすび

本稿では，命令シミュレーション手法をベースに実装した性能シミュレータ ESPRIT/sim を紹介した．

インタラクティブな分析に必要な高速性のため，複数のシミュレーション方式を併用し機能や精度とのトレードオフを実現した．高速動作に有効な動的バイナリ変換は最適化レベルを選択することにより開発費用を低減できる．また，C++言語を用いてプロセッサモデルをクラスライブラリ化することにより，異種 ISA のマルチプロセッサ動作を可能にし，派生クラスによりモデル作成の容易化を実現した．遅延要因を加算可能な単純なモデルとすることにより遅延要因の表現を直感的にした．HTML を用いた表とグラフ出力は，移植性とインストール容易性に優れかつユーザインターフェイスとして効果的である．

今後，概略サイクル数の見積り精度について評価をし，改良をする．

文　献

[1] C. Cifuentes, V Malhotra, "Binary Translation: Static, Dynamic, Retargetable?," pp.340-349, Proc. Int. Conf. On Software Maintenance, Monterey, CA, Nov. 1996

[2] B. Cmelik, D. Keppel, "Shade: A Fast Instruction Set Simulator for Execution Profiling," pp . 128-137, ACM SIGMETRICS, Nashviville, TN, 1994 (also available from http://portal.acm.org)

[3] M. Rosenblum, S. Herrod, E. Witchel, A.Guputa, M. Rosenblum, S. Herrod, E. Witchel, A.Guputa, "Complete Computer Simulation: The SimOS Approach", pp.34-43, IEEE Parallel and Distributed Technology, 1995.

[4] D. Burger, T.M. Austine, "The SimpleScalar tool set, version 2.0", Tech. Report 1342, Computer Science Department, University of Wisconsin-Madison, June 1997

[5] GNU "The GNU Profiler", http://www.gnu.org/software/binutils/manual/gprof-2.9.1/html_mono/gprof.html

[6] 平岡精一，近江谷麻人，西川浩司，山崎弘巳，"ソフトウェア資産活用に有効なバイナリトランスレーション技術" 三菱電機技報，vol.77, No7, pp.59-62, July 2003.

[7] M. Wakabayashi, H. Amano, "Environment for multiprocessor simulator development", Parallel Architectures, Algorithms and Networks, 2000. I-SPAN 2000 Proceedings, pp.64-71, International Symposium on 7-9 Dec. 2000

[8] 中田 尚，大野和彦，中島 浩，"高性能マイクロプロセッサの高速シミュレーション，" 先進的計算基盤システムシンポジウム SACSIS2003 論文集，pp.89-96,2003

[9] 吉瀬 謙二，片桐 孝弘，本多 弘樹，弓場 敏嗣，"SimCore/Alpha Functional Simulator の設計と実装，" 電子情報通信学会論文誌 D-I vol J8-D-I No.2 pp.143-154 2005

[10] "PowerPC™ Microprocessor Family: The Programming Environments for 32-bit Microprocessors", Motorola, 1997.

[11] ルネサステクノロジ(株)，"M32R ファミリ M32R/ECU シリーズ" ,http://japan.renesas.com/media/products/mpumcu/child_folder/04_m32r.pdf

[12] ルネサステクノロジ(株)，"SuperH RISC engine ファミリ" http://japan.renesas.com/media/products/mpumcu/child_folder/03_sh.pdf

[13] SPEC CINT95, http://www.spec.org/cpu95

[14] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A.Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," Proc. 22nd International Symposium on Computer Architecture, pp.24-36, June 1995.

[15] Simplescalar LLC, http://www.simplescalar.com/