計算機アーキテクチャ 119－20
（１９９６．８．２８）

SPEC92 命令トレースの生成と解析：PVS トレース機能拡張による

大津山　公平、Sergey V. Ten

会津大学コンピュータ理工学部
〒965-80 福島県会津若松市一箕町鶴賀
Tel:(0242)37-2578, Fax:(0242)37-2548
{kohei-o,ten}@u-aizu.ac.jp

小林　誠

Apple Computer Inc.
1 Infinite Loop,
Cupertino, CA 95014 U.S.A

あらまし：　パイプライン制御、スーパースカラ、分岐予測、多階層キャッシュなどを用いた最近の高速計算機の正確な性能評価を行うために命令トレース駆動シミュレーションはますます重要性を高めてきている。しかしながら命令トレースの作成は多大な労力を必要とする。今回、我々は PowerPC の仮想計算機環境である PVS に malloc などの基本関数を追加することで、命令トレースを容易に採集することを可能にした。本論文では PVS の機能拡張と SPEC92 トレースの解析結果について報告する。

キーワード：　命令トレース、PVS、命令トレース解析、プログラム動特性

Abstruct:　RTL(Register Transfer Level) Simulation using instruction traces become more and more important for evaluating performance of the high-performance processor which use technologies like super-scalar, branch prediction and multi-level cache. But, still it is usally a hard task to generate instruction traces.

In this paper, we report the extension of PVS(PowerPC Virtual System) to gather instruction traces easily and we have done some analysis on the traces we generated.

Keyword:　Instruction Trace, PVS, Instruction Trace Analysis, Dynamic behaviour of Program Execution

1 はじめに

プロセッサの性能評価を行う際には通常、解析的手法、シミュレーション、実機による実測などが行われるが、近年はプロセッサが高速化のためのパイプライン制御や、スーパースカラ、分岐予測、多階層キャッシュなどによってますます複雑化している。解析的手法ではワークロードのチューニング等によっても精度は高々10%程度と精度が不足し、一方、実機による実測では、実機の開発後に行われるので性能評価時期が遅すぎる。このため近年ではレジスタ転送レベル(RTL)でのシミュレーションが多く用いられるようになって来た。シミュレーションは（作りにもよるが）精度が比較的高く（最高1%以内）、時期的にも早い段階から性能評価作業に入ることが出来る。

シミュレーションのためのワークロードとして、命令MIX（つまり命令出現頻度）等が使用されたこともあるが、上述のような理由により最近では命令トレースが使用される。プロセッサの正確なモデルに対しては正確な命令トレースが要求され、正確な命令トレースは性能評価上ますます重要になってきている。

2 トレーサ

命令トレースはトレーサによって採取される。トレーサには大きく分けて、ハードウェア方式、ソフトウェア方式、ハイブリッド方式の3種類がある。

2.1 ハードウェア方式

カウンタやメモリ、付随する記憶装置等を対象システムに高インピーダンス・プローブにてハードウェア的に接続させ、情報を収集する。対象システムに影響を与えないので、非常に精度の高いトレースが採取出来る。通常はロジックアナライザを使用するが、市販のものでは記憶容量が少ないため、ロジックアナライザを改造して大容量メモリに接続し、十分長いトレースを生成するような研究や[7]、ロジックアナライザを使用せず、ハードウェアを自作した例[9]もある。

ハードウェア方式においてはオーバーヘッドはないという非常に大きな利点があるものの、トレーサーの費用は莫大なものになる。また、バス・インターフェースなどチップ外部の信号しか測定出来ないこと、タスクのディスパッチなどソフトウェア・イベントは測定出来ないという欠点を持つ。

2.2 ソフトウェア方式

ソフトウェア方式には大別すると、以下の2つの方法がある。

・トラップ（インライニング）

これはトラップ命令をオブジェクト又はソースコードに埋め込むもので、ソースやオブジェクトが入手可能なものに関しては有効である。トラップ命令が実行されると、トレース用のルーチンに入り、命令情報を採取する。オーバヘッドは実行されるトラップ命令の頻度によって異なってくる。

・トレースモード（シングルステップ）

いわゆる、シングルステップモードと呼ばれるモードで、1命令実行毎にトレース用のルーチンに割り込み、トレースを採取する。命令の解析をソフトウェアで行うためにオーバーヘッドが約3000倍以上と非常に実行時のオーバーヘッドが高い。

・プロセッサ・シミュレーション

レジスタ転送レベル(RTL)アーキテクチャシミュレーションによってプログラムを実行して、トレースを生成するものである。上2つの実機による方法に比べて更に実行時間がかかる。実現例としては今回我々が着目したPVSなどがある。[10]

2.3 ハイブリッド方式

ハードウェアによる方法ではマイクロプロセッサーの内部状態である命令実行トレースを取ることは出来ない。一方ソフトウェア・トレーサーではオーバーヘッドが大変大きいので測定対象のシステム挙動を変えてしまう。そこで、ソフトウェア・イベントも含めた命令トレースを少ないオーバーヘッドで生成出来るようにソフトウェアとハードウェアを両方使用したハイブリッド方式も用いられている。

さらに、最近では非侵略的(non-intrusive)なトレース方法が提案され既にある程度の成功を納めている。この方法ではメモリ・バス上のアドレスとデータをハードウェア・モニターを用いて実時間で記録し、後にこれらから命令トレースを生成するものである。

3 トレーサの問題点

さて、以上のように命令トレースを採集する方法は多くあり、以前から試みは数多く行われているが、トレーサには以下に挙げるような問題点がある。

・高価

特にハードウェア・トレーサは高価であるし、ソフトウェア・トレーサはソフトウェアの開発のためにかなりの工数がかかる。

・オーバヘッド

特にトレースモードによるソフトウェア・トレーサは非常にオーバーヘッドがかかる。また、シミュレーションの場合は実行時間がかかる。

・不正確

オーバーヘッドに付随する問題であるが、トレーサが動作するために、システムのタイミングが変わり、従って挙動が変わってしまい、正確なシステム・トレースがとれないことがある。例えばトレース中はタイマー割り込み等の外部割り込みの頻度が相対的に増加し、割り込み解析ルーチンの起動頻度が相対的に高くなってしまう。

・ＯＳトレース

特にソフトウェア・トレーサの場合はＯＳ部分、特に特権モードで走行するモジュールまでトレースをするのは困難である。また、シミュレーションの場合は、ＯＳカーネルのオブジェクトを用意する必要があり、これも通常は入手困難である。

4　ＰＶＳに対する機能拡張

今回我々が使用したのは、PVS (PowerPC Virtual System ; 以前は PowerPC Visual Simulator と呼ばれていた) という IBM 社によって作られた PowerPC 用のアーキテクチャ・シミュレータである。PVS には以下のような特長がある。

- IBM RS6000ワークステーション上で、オブジェクトコードによる PowerPC マイクロプロセッサのアーキテクチャシミュレーションを行うことが出来る。
- アーキテクチャ定義ファイルを追加することで、PowerPC シリーズの各プロセッサ (601,603,604,620 等) のシミュレーションが可能である。
- PVS には命令トレース採集機能があり、この機能によって命令トレースを非常に簡単に得ることが可能である。

しかしながら、PVS では通常のプログラム実行に必要な基本的な関数を備えておらず、実行出来るプログラムが非常に限られており、従ってベンチマークなどの実際のプログラムのトレースはほとんど不可能であった。

今回我々は C 言語で使用する全 I/O 関数と動的メモリ割り当て関連関数、乱数発生関数を付加して PVS の機能を拡張することにより、より多数のプログラムに対応し、容易に命令トレースを採取できるようにした。以下に機能拡張の詳細を説明する。

4.1　PVS の動作

PVS は１つのプロセス上にある仮想計算機システムであり、独自のメモリ空間を持っている。このメモリ上にユーザプログラムや、プログラムにリンクされているライブラリ等が存在する。また、別のプロセスで I/O model が実行されており、両者は API(Application Interface)同士でプロセス間通信を行い、PVS の主記憶中のバッファ領域のデータを介してデータのやりとりを行う。I/O modelからは実際の I/O 命令が発行される。

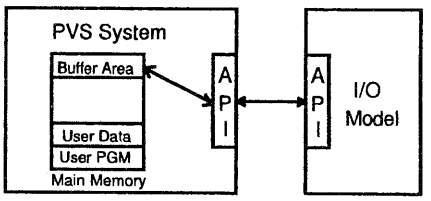

図１　ＰＶＳ構成図

4.2　I/O 関連関数

fopen()、printf() 等の基本的な I/O 関数は IBM からソースの形で提供されている iolib.c に収録されている。我々は実現されていない以下の関数を同様のインタフェースで実装した。

```
fprintf(),fscanf(),putc(),
getc(),fgets()
```

また、stdin、stdout、stderr に対する特別な処理（事前のファイル・オープンなど）を追加した。これらの修正は全て iolib.c に対して行われた。このプログラムはユーザプログラムのコンパイル時に静的にリンクされることによってユーザに I/O 機能を提供する。

4.3　動的メモリ割り当て関連関数

malloc()、realloc()、free() は I/O と同じインターフェースを使用して、全く新規に実装された。これらの関数はバッファエリア内に設けられた256MBの領域を使用し、メモリの割り当てを行う。

4.4 その他の拡張

その他の拡張としては、以下の乱数を発生させる関数を実装した。

```
srand(),drand(),rand(),srand48()
```

4.5 Fortran 関連

FORTRANのプログラムは直接これらI/O関数を呼ぶことが出来ないので、今回はソース中の簡単なREAD/WRITE文だけを自分で作成したC関数を呼ぶように変更して対処した。

5 トレース採取・処理例

以上の機能拡張を加えたPVSシステムによって、SPEC92ベンチマークのうちの3本のプログラムの命令トレースを採取することが出来た。（容量の関係で、命令数は100万命令前後とした）以下にトレースの一覧を示す。なお、各ベンチマークともに最初から約百万命令実行したものと、その後二百万命令スキップしてから約百万命令実行した2つのトレースを採取し、それぞれ、例えばespressoならesp0とesp1というように命名した。

表1 トレース諸元

トレース名	esp0	esp1	fpp0	fpp1	tom0	tom1
種類	INT92	INT92	FP92	FP92	FP92	FP92
出典	espresso	espresso	fpppp	fpppp	tomcatv	tomcatv
言語	C	C	FORTRAN	FORTRAN	FORTRAN	FORTRAN
命令数	962,947	999,999	1,000,000	1,000,000	1,000,000	1,000,000
BasicBlock数	369	454	389	265	21	5
平均命令長	6.91	8.58	16.73	35.21	25.19	36.80

以下ではこれらのトレースの特徴を知るために2つの処理を行ってみる。

5.1 データ可視化

得られたトレースデータを使用して行った一番目の処理はデータの可視化である。ワークロードの特徴を可視化するには、何らかの処理を行い、グラフ化するのが一般的であるが、ここでは、命令アドレスとオペランドデータアドレスの動きを見るために、トレースデータから命令アドレスとオペランドアドレスを抽出し、命令の順番を横軸に、命令とデータのアドレスを縦軸にしてプロットした。ここではespressoとfppppを見てみることにする。

なお、元のグラフはカラーであるが、白黒印刷である関係上、命令とデータの区別がつき難くなっている。だいたい、命令は低いアドレスを、データは高いアドレスを占めている。

(1) espresso

espressoのトレースesp0、esp1に対応するグラフは以下のようになる。最初の百万命令(esp0)ではアクセスパターンにほとんど変わりがなく、同じような命令が実行され、それに従って同じようなデータがアクセスされている。途中の百万命令(esp1)では命令、データともにかなり動きがあることが分かる。

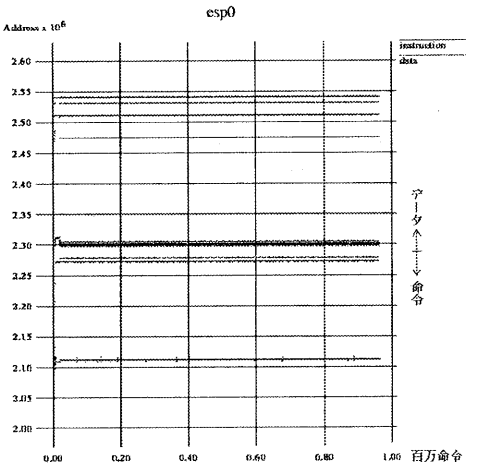

図2 esp0アドレス散布図

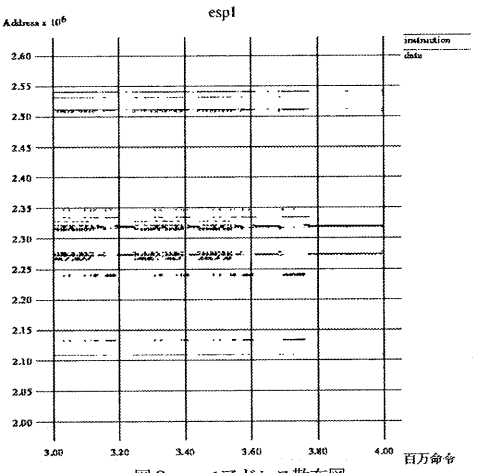

図3 esp1アドレス散布図

(2)fpppp

最初の20万命令ぐらいまではデータの初期化のためのルーチンによりデータのアクセスパターンが以後と異なっている。しかしその後は2つのトレースを通して同じようなパターンが続いており、同じようなループを実行していると考えられる。

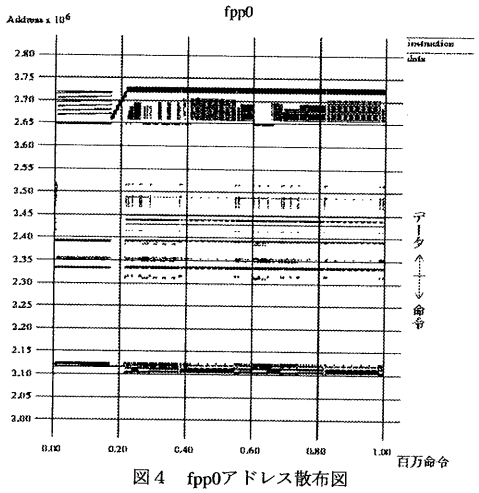

図４　fpp0アドレス散布図

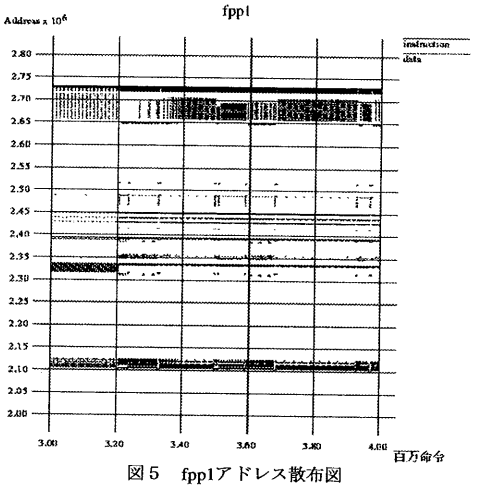

図５　fpp1アドレス散布図

5.2　ベーシックブロック分析

次に、命令とベーシックブロックの出現頻度を調査した。これは会津大学で行われたオブジェクトコードによるワークロードの特徴化（static analysisと呼んでいる）[6]を命令トレースにも当てはめたもので、我々はDynamic Analysisと呼んでいる。

(1)命令出現頻度

横軸に命令を使用された頻度順にとり、縦軸に累積出現頻度をプロットした。

espresso は20命令で、fppppは40命令で、tomcatvでは10命令で出現頻度はほぼ100%に達する。グラフの立ち上がりが極めて急峻であることはtomcatvの際だった特徴と言えるであろう。更に細かく見て行くと、どのベンチマークのトレースでも最初より途中のものの方が傾きが急峻である。これは、途中のトレースはループの最中である確率が高く、使用する命令が比較的限られているからだと考えられる。

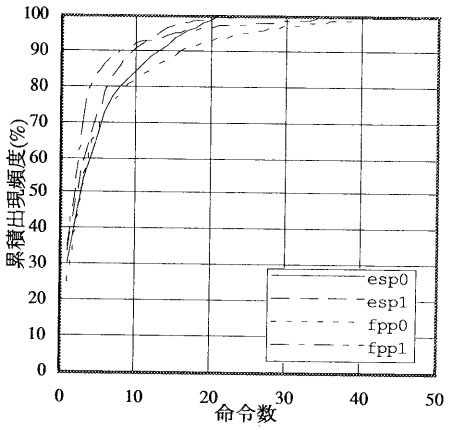

図６　命令出現頻度(esp0,esp1,fpp0,fpp1)

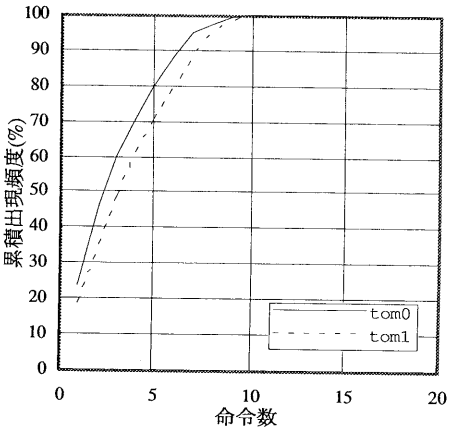

図７　命令出現頻度(tom0,tom1)

(2)Basic Block出現頻度

まず、Basic Blockにおいて、(使用された頻度×Basic Block 内の命令数)を重み付き頻度と呼ぶことにする。横軸に Basic Block を重み付き頻度順にとり、縦軸に累積重み付き出現頻度をプロットした。[11]

espresso、fpppp は割と似たようなパターンを示しているが、tomcatv は Basic Block の総数からしても(esp0が21、esp1が５）極端に少なく、２つの Basic Block でほぼ100%になってしまうことが分かる。

また、Basic Block の平均長を表１に記載したが、平均長は最初より途中のものの方が長い傾向にある。これも、途中のトレースは大きなループの最中であることを示唆している。

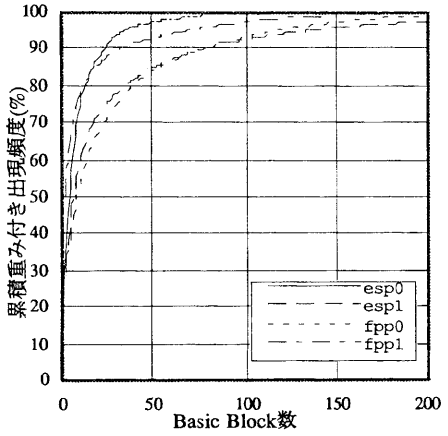

図8　Basic Block出現頻度(esp0,esp1,fpp0,fpp1)

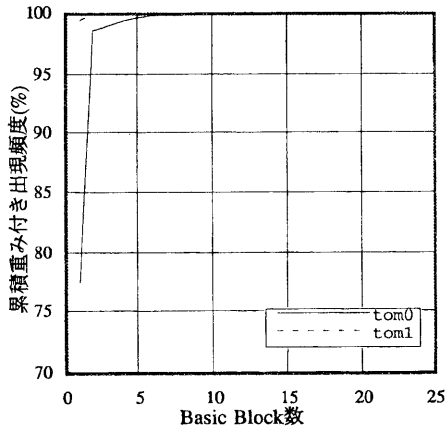

図9　Basic Block出現頻度(tom0,tom1)

6　終わりに

今回我々はPVSにI/O機能とmalloc機能を付加し、SPEC92ベンチマークのうち数プログラムのトレースを採取することが出来た。このインプリメントはC関数に対して行ったが、Fortranプログラムでも READ/WRITEだけをC関数を呼ぶようする若干のソース修正で対応が可能である。また、採取したトレースを基に、トレースの可視化とDynamic Analysisという2つの処理方法の紹介を行った。

Cプログラムでトレース採取に成功していないものは、fork()、またはsignal()という未実装な関数を使用している。今後はこれら未実装の関数を実装してSPEC92ベンチマークの全トレースを作成して行きたい。Fortranに関しては、まだ入出力が非常に単純なものだけしかトレース作成を行っていない。今後はFortranのルーチンも作成し、ソースの変更なく実行を可能にして行きたい。

シミュレーションによるトレース作成ではカーネル自体も実装することで、カーネル内のトレースも作成出来るようになる。AIXのソースコードがIBMより入手でき次第カーネルの実装を行いシステム全体のトレースの採取に着手することを計画している。

7　参考文献

[1] C. Stephens, B. Cogswell, J. Heinlein, and G Palmer, "Instruction Level Profiling and Evaluation of the IBM RS/6000", *Proc. of 18th Int. Symp. on Computer Architecture* , ACM, 1990, pp180-189

[2] "SPEC Newsletter", Vols 1-7, Standard Performance Evaluation Corporation, 1989-1995

[3] R. Giladi, N. Ahituv, "SPEC as a Performance Evaluation Measure", *Computer*, IEEE, August, 1995, pp33-42

[4] J. R. Larus, "Efficient Program Tracing", *Computer*, IEEE, May, 1993, pp52-61

[5] T.A. Diep, C. Nelson, and J.P.Shen, "Performance Evaluation of the PowerPC 620 Microarchitecture", *Proc. of 22nd Int. Symp. on Computer Architecture* , ACM, 1995, pp163-174

[6] Makoto Kobayashi and Sergey Ten, "Static analysis of Common Instruction Sequences: SPEC Benchmarks",University of Aizu Technical Report 95-1-034, October,1995

[7] Takashi Horikawa, "TOPAZ: Hardware-Tracer Based Computer Performance Measurement and Evaluation System", *NEC Research & Development* , Vol. 33, NEC, October 1992, pp638-647

[8] Domenico Ferrari, "Computer Systems Performance Evaluation", Prentice Hall, 1978

[9] K.Flanagan, K. Grimsrud, J. Archibald, B. Nelson, "BACH: BYU Address Collection Hardware", BYU Technical Report TR-A150-92.1, 1992

[10] "PowerPC Visual Simulator User's Guide", Version 2.2, IBM Corporation, 1995

[11] M. Kobayashi, "Dynamic Profile of Instruction Sequences for the IBM System/370", IEEE Transaction on Computers, Vol.32, No.9, September, 1983, pp859-861