計算機アーキテクチャ　148－1
（2002.　5.　13）

高頻度実行部の抽出による大規模トレースの縮小

上田　晴康[†]　安里　彰[†]

性能評価をするために、サイズの大きいベンチマークプログラムの全実行トレースをトレースベースのシミュレータに適用すると、莫大な量のシミュレーションが必要であり、現実的ではない。本論文では、全体のトレースから多数の短いトレースを抽出することで、短時間に精度の高い性能評価を行う手法を示す。本手法は、トレース中の高頻度部分の抽出、キャッシュのコールドミスの補正、重み付き集計の 3 段階から成る。本手法を用いて中間的な評価を行うため、CFP2000 ベンチマーク中 172.mgrid から抽出したトレースと実機での性能を比較した。オリジナルのプログラムの 300 分の 1 のトレースの評価を集計したところ、トレースのコールドミスの影響がかなり強く出る結果となった。

Extracting highly frequent trace to evaluate huge benchmark test

Haruyasu Ueda[†] and Akira Asato [†]

Trace-driven simulator is useful tool to evaluate architecutre. But, with huge benchmark test, it is infeasible to apply whole trace of the test to simulator because the trace size is huge. We propose a method to extract short traces from whole huge trace. The method consists of extraction of highly frequent trace, correction of cold cache miss, and aggregation of each extracted trace. We applied this method to 172.mgrid test in SPEC CFP2000. The total length of extracted trace is 1/300 of whole trace but the characteristics of extracted traces show strong cold miss effect.

1. はじめに

コンピュータシステムの開発において、迅速かつ正確な性能予測・評価は重要な課題であるが、最近の開発期間の短縮化に伴ってその重要性はさらに増しつつある。その一方でコンピュータシステムの速度向上に伴い、速度指標となるテストプログラム（ベンチマークテスト）は使用メモリ量や実行時間が巨大化する傾向にあり、実機が完成する前にこれらのプログラムを直接用いた性能予測を行うことが困難になってきている。この問題に対処するため、我々は評価対象とするベンチマークテストの特性を保持しつつ、評価にかかる時間および使用メモリを縮小する研究を行っている。

ベンチマークテストプログラムの縮小に関連した研究としては、ソースコードや入力データの変更によるワークロード縮小の研究と自動的にトレースの縮小を行う研究がある。

ソースコード変更によるワークロード縮小は、本論文とは別に我々が提案した手法であるが [6),7)]、全体の実行の特性（CPI、キャッシュミス）を反映した小さな実行プログラムを作ることでプログラム縮小を試みている。この手法は比較的小さなプログラムで精度の高い性能評価が出来、トレースを保存するための多量の領域を必要としないという特徴がある。また類似研究として入力データの変更による計算量の縮小の研究も行われている [14)]。これらの手法は、プログラムやデータの作成に多くの手間がかかるという問題点があり、自動的な縮小作業が必要とされている。

一方、自動的に縮小する方法としては、サンプリング抽出や類似する命令列の抽出、プロファイルを用いたトレース作成などの研究が行われている。

サンプリング抽出による方法 [1)] は単純であるが、サンプリングの偏りが無視できるようにするために相対的に多くのサンプルを必要とし、ある程度の重複した評価は避けられない。また、取るべきサンプルの数はベンチマークテストの性質に依存しており一概に決めることは出来ない。例えば、数値シミュレーションや科学技術計算では高度に繰り返し計算が多いので少数のサンプルで十分であるが、コンパイラ、ファイル圧縮など整数計算が中心のベンチマークテストでは、多くの異なるフェーズが混在しており、比較的多くのサンプルを必要とする。このため、十分な数のサンプルを取ったかどうかを事後に検証するのが難しく、必然的にサンプル数が十分でない場合のデータの信頼性は低くなる。

トレースファイル群より代表トレースを選出する研究も行われている [2)~4)]。これらの手法は数多くのト

† (株) 富士通研究所
FUJITSU LABORATORIES LTD.

レースから最も典型的なトレースを選択する。これにより重複するトレースを排除することが可能である。しかし、いくつサンプルを残すかということや、サンプルトレースのサイズをどのくらいにするかということは経験的に決められていることが多い。

またプロファイルを用いてトレースを作成する方法 [5] は、あらかじめプロファイラで得た基本ブロックの実行回数に応じてトレースをつくり出す手法で、小さなトレースファイルから精度の高い性能評価が出来る。

本論文では、プログラムの実行トレースを解析することで、高頻度実行部のトレースを抽出し、重複の少ないトレースから元のプログラムの性能評価を行う。この手法はプロファイルを用いるトレース作成に近いが、プロファイラを介さず、直接フルトレースを解析してトレースを抽出する点が異なる。また、キャッシュのコールドミスの影響を相殺するために複数のトレース情報を用いることが出来ることと、複数の小さなトレースを用いるため、評価を並列に行うことができるよう拡張されていることが特徴である。

本論文では、まず 2 節で高頻度トレース抽出による縮小手法について説明し、次にこの手法を用いた SPEC CFP2000[8],[9] のうちの 171.mgird ベンチマークに関してトレース抽出した時の中間的な実験結果と考察を 3 節に示す。

2. 高頻度トレース抽出手法の概要

実行トレースを解析して高頻度実行部のトレースを取り出すには、1) 実行トレースの作成、2) トレース中で同じ命令列を繰り返し実行している部分の検出、3) 繰り返し部分のうち全トレースに占める割合の大きい部分の抽出、の 3 つのステップが必要である。そして、得られた部分トレースを用いて評価を行うには、4) 繰り返し回数による重み付き合計による評価結果の集計が必要である。これらのステップを図 1 に示した。

以下に詳細を述べる。

2.1 実行トレースの作成

実行トレースとは、対象となるプログラムが実行を行う際に、命令毎にどのような動作をしたかを記述したものである。トレースの採取には実行トレーサと呼ばれる特別なプログラムを用い、命令のシミュレーション等を行ってトレース情報を採取する。本論文の評価では、Sparc アーキテクチャ [10] の計算機を使用したため、shade[11] を利用してトレース情報を採取した。shade のトレース情報は、実行命令のアドレス、実行した命令語、命令中でメモリにデータアクセスをした場合はその参照アドレス、分岐方向などのフラグから構成されている。

長時間の実行に対するトレースは、圧縮しても莫大な量になることが多い。そのため作成したトレースは

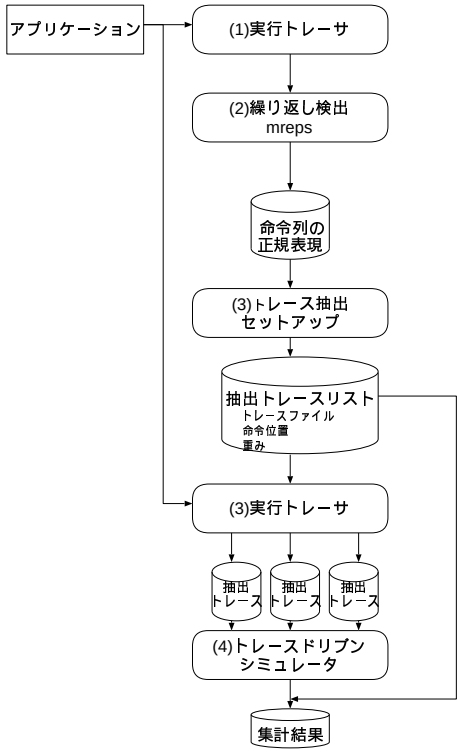

図 1 高頻度トレース抽出手法の構成

直接パイプ等で繰り返し部分の検出に渡して処理を行う。

2.2 繰り返し部分の検出

本論文では mreps[12] を用いて繰り返し部分を検出している。mreps は文字列中で 2 回以上繰り返す部分を入力の線形時間で検出する、高速のアルゴリズムを実装したものである。このアルゴリズムの概要は付録に添付した。mreps を用いることで、実行トレース中に存在するさまざまなサイズのループを効率良く検出することが出来る。公開されている mreps プログラム (ver. 1.1) は検出の単位が 1byte 単位であるため、本論文では複数 byte を 1 単位とした検出が出来るように拡張して繰り返し部分の検出を行った。尚、繰り返し部分の検出の際にトレースの実効命令が同じかどうかを比較する必要がある。本論文ではインライン展開などで同じ命令語が異なるアドレスに配置された場合を考慮して、実行した命令語のみを比較対象として繰り返し部分の検出を行った。

mreps の出力には以下にに示す問題点があるため、直接実行トレースの検出に用いることは出来ない。一つは、mpreps は多重ループを検出すると、内側のループと外側のループをそれぞれ独立のループとして扱い出力を行う。しかし、実行トレース中の繰り返し検出の観点からすると、多重ループは外側のループと内側のループを関連しあったものとして扱いたいというこ

```
(dd38bff8c568a0b0dd38a0008e01e008
(80a5200884012001){123}
dd38bff8c568a0b0){124}
```

図 2 繰り返し検出の出力例 (正規表現)
16 進の 8 文字 (32bit) が 1 命令である。この出力例は $(4+2*123+2)*124 = 31248$ 命令の命令列を示している。

とである。

また、mreps は入力を一定サイズのバッファ毎に処理しており、原理的にバッファサイズより大きな周期を持つ繰り返しを検出することは出来ないのでまた、バッファサイズより小さいループでも、ループは始まりや終りが欠けてしまうことがある。一方で、mreps はバッファサイズを大きくすると非線形に実行時間が長くなる性質を持つため、あまり大きなバッファサイズで mreps を実行することも出来ない。

そこで、mreps の出力を加工して、多重ループの検出を行い、バッファ程度のサイズのループで、バッファ境界によって欠けてしまった始まりと終りを結合し、再度 mreps にかけられるよう、正規表現の形で出力を行うプログラムを作成した。出力の正規表現は perl の物を用いている。図 2 に出力の例を示す。

このプログラムを介して mreps を 2 回適用することで、莫大な (数百 G 命令分の) トレースから繰り返し部分を削除して正規表現として表現された命令列を得ることができる。

2.3 繰り返し部分の抽出

mreps による繰り返し部分の検出の出力は、実行命令列に関する繰り返し回数の明示された正規表現として得られる。この出力から実行命令の長さが非常に長い部分は容易に見つけることが出来る。しかし、ここで得られる情報は実行命令のみで、実行アドレス、データ参照アドレス、フラグなどの情報は欠落している。なぜならば、二度目に mreps を使う際に、命令列のみを用いて同一の繰り返しであることを検出するために、実行アドレス、データ参照アドレスなどは、出力から除かれるためである。この結果、そのままではデータ参照アドレスを必要とする通常のトレース解析ツールは使うことができない。

そこで、検出された長い繰り返し部分から実行された命令数の範囲を計算して、再度実行トレースを作成し、次に計算された命令数の範囲の部分のみを保存し、抽出トレースとした。保存が必要な部分は、厳密には評価方法によってやや異なるので、詳細は次節に示した。抽出されたトレースは、キャッシュシミュレータやアーキテクチャシミュレータを用いて性能評価に使うことができる。また、各抽出トレースは全実行時の繰り返し回数と関連付けられており、集計の際に重み付けすることができる。

2.4 評価結果の集計

抽出されたトレースは各種のシミュレータを用いて性能評価指標を測定することができる。しかし、抽出すべきトレース部分は評価指標によって多少異なる。

2.4.1 キャッシュミスが重要でない場合

もっとも単純な場合、例えば命令シミュレータでキャッシュを含まない CPI を計算する場合には、高頻度で発生する命令列のうち、最内ループの 1 周分を抽出すればよい。抽出された各トレースに関して命令数と実行サイクル数をシミュレーションで得、各抽出トレースの繰り返し回数で重み付けした総和をプログラムの全実行の命令数とサイクル数として評価を行うことができる。

2.4.2 キャッシュミスが重要な場合

一方キャッシュミスが重要な場合、コールドキャッシュミスに対する補正が必要となる。コールドキャッシュミスとは、命令実行時にキャッシュに何も入っていないためにおきるキャッシュミスである。通常の実行を行っている場合には、キャッシュ中に命令やデータが入っている事が多い。このような命令に関するトレースのみを部分トレースとして抽出すると、抽出した部分トレースはコールドキャッシュミスを起こし、評価が不正確になる。

コールドキャッシュミスに対する補正を行うためには、キャッシュをウォームアップするためのトレースを抽出すべきトレースに追加する。ウォームアップ部分を追加した部分トレース ($Warmup + x$) とウォームアップ部分のみからなる部分トレース ($Warmup$) の両方を用いて評価を行い、差分を取ることで抽出すべき部分トレースの純粋な評価結果が得られる。評価指標が $Perf(trace)$ と表現できるとすると、

$$(\text{真の})Perf(x) = Perf(Warmup + x) - Perf(Warmup) \quad (1)$$

この評価指標としては、命令数が増加するにつれ単調に増えるような指標を用いることができる。例えば、キャッシュミス回数やキャッシュミスペナルティを含んだサイクル数は式 1 で算出することができる。

さらにこの方法は、ループの 1 回目でキャッシュにデータが満たされ 2 回目以降はキャッシュミスが起きないような場合の補正にも用いることができる。この結果、ウォームアップの効果と 1 回目のループの効果を考慮したループ全体の評価指標の値は、式 2 で算出することが出来る。

$$\begin{aligned}(\text{真の})\ & Perf(n*x) = \\ & n*Perf(Warmup+x+x) \\ & +(1-n)*Perf(Warmup+x) \\ & -Perf(Warmup)\end{aligned} \quad (2)$$

FORTRAN で書かれた科学技術計算のように、ループ内のメモリアクセスがほとんど均一で、ループ 2 回目のキャッシュミス率とループ 3 回目以降のキャッシュミス率がほとんど変わらない場合は、この式によって

正確に評価を行うことができる。

3. 評　　価

本手法の有効性を確認するために、SPEC ベンチマークテストよりトレース抽出を試みた。対象ベンチマークテストは、ソースコード変更による縮小ワークロード [6] と比較するため SPEC CFP2000 より 181.mgrid を採用した。

評価作業にかかる時間が主な原因で、今回全トレースからの抽出は行わなかった。その代わり、均等に 30 個のサンプリングを行い、それらのサンプルが全体のトレースの 1%程度になるようにした。このサンプルトレースから抽出を行った。

抽出されたトレースは、キャッシュミス率が重要なケースを試みるため、主なループに関して、ウォームアップのみ、ウォームアップ+ループの 1 周目、ウォームアップから 2 周目までの 3 種を採取した。ループ選択の基準は、ループ全体の実行命令数が大きいものから順に選び、選択されたループの全実行命令数が全トレースの 9 割になったところで抽出を打ち切り、残りのループおよびループ以外のトレースは無視した。多重ループに関しては、最外周ループのみを取り出した。ウォームアップのためのトレースは一律 5M 命令とした。

評価指標としてはキャッシュミス率とキャッシュミスペナルティを含んだ CPI とした。トレースドリブンシミュレータとしては Paratool[13] を用いた。測定に当たって、アーキテクチャパラメータは [6] で評価対象として用いられている実機 (Fujitsu PRIME-POWER GP7000F) と同じ構成を選んだ (表 1)。

fetch 幅/issue 幅	4/4
機能ユニット	IALU:4 LD/ST:2 FADD:1 FMA:1
命令 1 次キャッシュ	128K ダイレクトマップ
データ 1 次キャッシュ	128K ダイレクトマップ
2 次キャッシュ	2M ダイレクトマップ

表 1　対象アーキテクチャパラメタ

3.1 実 験 結 果

Paratool よりサイクル数および 2 次キャッシュミス数を求め、式 2 に従って重み付けした後 CPI および 2 次キャッシュミス率を求めた結果を、表 2 に示した。重み付けした結果、3 本の抽出トレースにつき一つの評価指標が得られる。

ここに示した実行命令数は、検出したループの総命令数で、ウォームアップ分を含まない。すなわち、抽出トレースによってカバーされた命令数である。

3.2 抽出トレース

抽出されたトレースの統計情報を表 3 に示す。

	実行命令数	CPI	2 次キャッシュミス (%)
抽出トレース合計	4.47G	0.715	1.11
オリジナル	464G	1.038	0.8696
ソースコード変更 [6] による縮小	138M	-	0.6597

表 2　Paratool による実験結果

検出ループ数	127
総トレース個数	381
総命令数	1.73G
うちウォームアップ	1.57G
非ウォームアップ	160M
オリジナル総命令数	464G
ソースコード変更 [6] による縮小命令数	138M

表 3　抽出されたトレースの統計情報

3.3 考　　察

実験結果は、かなり精度の低い評価しか出来なかったことを示している。特にキャッシュミスが非常に高く出ており、その影響で CPI が低くなっている。

原因の一つは、全トレースから抽出せずに、サンプリングをしてしまったためと考えられる。サンプリングを行うと、全体として大きなループと見なされる部分が、それぞれ別のループであるように認識されてしまう。このため、本来長いループの途中で、キャッシュミス無しで実行できていた命令がキャッシュミスが起きたものとして扱われてしまう。

抽出されたトレースサイズも期待される程小さくない。この主な原因はコールドキャッシュミスを防ぐためのウォームアップ分である。ウォームアップ分を除けばオリジナルの、約 1/4000 となり、ソースプログラム修正で得られる実行命令数に匹敵する。

一つの解決法はトレース抽出を 2 ステップに分けることである。最初のステップではキャッシュシミュレーションだけを行い、ウォームアップが必要かどうかを調べる。次のステップでは、不要なウォームアップを除いたトレースを抽出し、より精度の高いが時間のかかるシミュレーションを行うことである。

4. お わ り に

本論文ではプログラムの全実行トレースを解析して繰り返しを検出し、重複の少ないトレースを抽出する手法について述べた。この手法では mreps を繰り返し部分検出に用いており、莫大なトレースデータから実用的な時間内にトレース抽出が行える。

得られたトレースを集計した評価結果を実機の実行

結果と比較したところ、あまり高い精度が得られていないことがわかった。

今後の課題としては、より高い精度が得られるよう、サンプリングを用いないトレース抽出を試みる必要がある。また、より多くのプログラムで実験して、本方式の有効性を調査することがあげられる。その第一歩としてSPEC CPU2000の全ベンチマークテストでのトレース抽出を試みるべきであろう。その際に問題となるのは、以下の2点である。1) mrepsは完全に一致するループのみを検出する。このため、ループの内側に条件分岐があるようなループはループ毎に命令列が変わるため検出できない。Whole Path Profiling[15]と呼ばれる手法はこのようなループを含むトレースも扱えるので、応用出来る可能性がある。2) メモリの間接参照を行うようなループではコールドミスの補正がうまく働かない。この点に関しては、全トレース採取時にトレースの各命令がキャッシュミスを起こしたかどうかを調べることで、より精密な補正が可能となる。

参 考 文 献

1) S. Laha, J. Patel, and R. Iyer, "Accurate lowcost methods for performance evaluation of cache memory systems", *IEEE Trans. Comput., 37(11):1325-1336*, 1998.
2) Y.S. Iyengar, L.H. Trevillyan, and P. Bose. "Representative traces for processor models with infinite cache", *Proc. of the Second International Symposium on High Performance Computer Architecture*, Feb 1996.
3) T. Lafage, A. Seznec. "Choosing Representative Slices of Program Execution for Microarchitecture Simulations: A Preliminary Application to the Data Strem", *Workshop on Workload Characterization (WWC2000)*, Sep 2000.
4) K. Skadron, P.S. Ahuja, M. Martonosi, and D.W.Clark, "Selecting a Single, Representative Sample for Accurate Simulation of SPECint Benchmarks", *Tech Report TR-595-99*, Princeton Dept. of Computer Science, Jan. 1999.
5) P.K. Dubey and R. Nair, "Profile-driven sampled trace generation", *Technical Report RC 20041*, IBM Research Division, April 1995.
6) 稲田, 河場, 安里. "SPEC CFP2000 ベンチマークの縮小プログラム開発手法とその評価", 情報処理学会研究会報告 2002-EVA-2, Feb. 2002.
7) 小野寺, 上田, 安里. "SPEC CINT2000(181.mcf) の縮小プログラム開発手法とその評価", 情報処理学会研究会報告 2002-EVA-2, Feb. 2002.
8) J.L. Henning. "CPU2000: Measuring CPU Performance in the New Millennium", *IEEE Computer, 33(7):28-35*, Jul 2000.
9) "SPEC CPU2000 Press Release FAQ", available at `http://www.spec.org/osg/cpu2000/press`
10) "64 ビット RISC プロセッサ：SPARC64 GP", 雑誌富士通 2000-7, 2000. available at `http://magazine.fujitsu.com/vol51-4/paper06.pdf`
11) "Shade and Spixtools", available at `http://www.sun.com/microelectronics/shade`
12) R. Kolpakov and G. Kucherov. "Finding Maximal Repetitions in a Word in Linear Time", FOCS99, 1999. source file available from `http://www.loria.fr/~kucherov/SOFTWARE/MREPS/`
13) 志村 浩也他. "スーパスカラプロセッサの性能評価 – Paratool –", 情報処理学会研究会報告 93-ARC-102-1, Oct. 1993.
14) A.J. KleinOsowski, J. Flynn, N. Meares and D.J. Lilja, "Adapting the SPEC 2000 Benchmark Suite for Simulation-Based Computer Architecture Research", *Workshop on Workload Characterization, International Conference on Computer Design*, Austin, TX, Spt. 2000.
15) J.R. Larus. "Whole Program Paths", *Proc. of the SIPLAN '99 Conference on Programming Languages Design and Implementation (PLDI 99)*, May 1999.
16) M.G. Main. "Detecting leftmost maximal periodicities", *Discrete Applied Mathematics, 25:145-153*, 1989.
17) E.M. McCreight. "A space-economical suffix tree construction algorithm", *Journal of the ACM, 23(2):262-272*, 1976.
18) M. Crochemore. "An optimal algorithm for computing the repetions in a word", *Information Processing Letters, 12:244-250*, 1981.

付　　録

A.1 mreps のアルゴリズムについて

mrepsはMainによる最左最大繰り返し検出アルゴリズム[16]を応用したものである。このアルゴリズムは、まず入力から最左文字列を示す構造体suffix treeを作る[17]。次にそれを用いて入力文字列を部分文字列に分解して、sfactor（またはLampel-Zip decomposition) を作成する[18]。sfactorの構成に当たっては、入力文字列中のあらゆる2回以上の繰り返しに対して、その繰り返しの右端がsfactor部分文字列の右端となるように構成する。また、繰り返しが存在しない部分に関しては1文字ごとにsfactor部分文字列が構成する。

このように構成されたsfactorでは、繰り返し文字列の存在範囲がsfactor部分文字列の長さで限定できるため、全ての繰り返しを線形時間で検出できること

が証明されている [12)]。