計算機アーキテクチャ 147－22
ハイパフォーマンス
コンピューティング 89－22
（2002．3．8）

リバーストレーサによる性能評価用ワークロード生成

河場基行† 上田晴康† 安里彰†

CPU2000に代表される大規模ベンチマークテストの評価時間を短縮するため，性能テストプログラム作成システム***GREPT***を作成した．これはトレースサンプリング・リバーストレーサ(RT-lite)によって小さなテストプログラム(RTバイナリ)群を生成し，さらにバイナリイメージの差を用いたクラスタリングによって高速に代表RTバイナリを選出し評価対象を大幅に減らすことを特長としている．

CPU2000の8本のプログラムに対し，代表RTバイナリによるCPI推定を行ったところ，クラスタリング方式および代表RTバイナリ選出方式に改善の余地があるものの7.7%の誤差でCPIを推定できた．

Performance Test Program Generation Using the Reverse Tracer

MOTOYUKI KAWABA,† HARUYASU UEDA† and AKIRA ASATO†

We describe a performance test program generator, called ***GREPT***, developed to shorten the evaluation time for large benchmark sets such as CPU2000 benchmark. The ***GREPT*** can generate several small representative test programs by trace sampling, reverse tracing, and clustering. For the speed of the clustering, our clustering method uses the simple comparison between binary images of test programs.

Our preliminary experiments show the CPI estimation using representative test programs has 7.7% errors on average for 8 programs from CPU2000 benchmark, though the quality of clustering and the representativeness of selected programs still have the room of the improvement.

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あいまって，速度指標となるテストプログラム(ベンチマークテスト)も使用メモリ量や実行時間の面で巨大化しており，実機が完成する前に性能予測そのもの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プログラムを空間的・時間的に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しては数多くの研究がなされている．代表的なものとしてトレースサンプリング技術[1]がある．これは一定あるいはランダムな間隔でトレースを採取し，性能解析を行う技術である．採取されたサンプルトレースが全体の実行を反映していることが前提になっている．この手法はサンプル数が多いほど評価精度が向上するが，これに伴いトレース解析時間が長くなってしまう欠点がある．そこでトレースファイル群より代表トレースを選出する研究[2)3)]も行われている．これらは主にトレースデータの縮小を目的とした研究である．

我々は縮小ワークロードを実行バイナリ形式で生成する技術に着目している．実行バイナリ形式で生成すると適用範囲が広がるという利点がある．例えば，命令トレーサを用いたソフトウェアシミュレーション評価が可能であるとともに，論理シミュレータによる詳細な評価にも適用できる．この特長を利用してシミュレーションモデル検証を行うことも可能である．我々は縮小ワークロードを実行バイナリで生成する手法として，プログラムのソースコード変更によるワークロード縮小手法を提案した[8)9)]．この中で全体の実行の特性(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．ところがソースコード変更によるワークロード縮小手法は，一般プログラムに適用することはかなり難しい．

そこで一般アプリケーションにも適用できる性能テストプログラム作成システム***GREPT***☆を作成した．***GREPT***ではトレースサンプリング技術，リバース

† (株)富士通研究所
FUJITSU LABORATORIES LTD.

☆ Generating REpresentative Performance Tests

トレーサ技術[4]，およびクラスタリング[5]による代表プログラム選出技術を用いて小さな性能評価用テストプログラム群を作成する．クラスタリングにあたっては，バイナリを直接比較することで代表プログラム選出時間の短縮を実現した．

本論文ではまず第2節でSPARCアーキテクチャ[11]用に実装した***GREPT***の概要を説明する．次に第3節でCPU2000[6]プログラムを対象に***GREPT***を適用した実験結果を示す．最後に第4節で考察を述べる．

2. *GREPT* 概要

2.1 処理の流れ

GREPTの概要を図1に示す．***GREPT***はサンプリングトレースを採取するトレースサンプラと，テストプログラム群(RTバイナリ群)を生成するRT-liteと代表プログラム(代表RTバイナリ)を選出するREPTS☆から構成される．***GREPT***では以下のように代表RTバイナリ群を生成する．

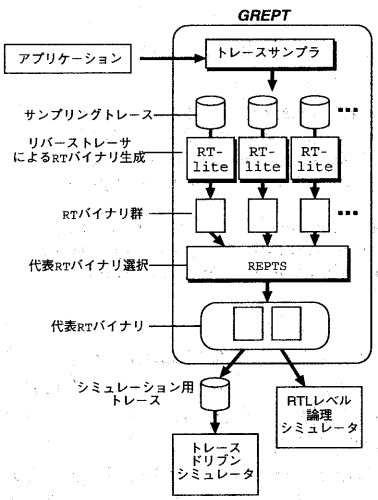

図1 性能テストプログラム生成システム***GREPT***

(1) トレースサンプラを用いてランダムな命令間隔でサンプリングトレースを生成する．

(2) リバーストレーサRT-liteを用いて，RTバイナリ群を生成する．

(3) REPTSによってRTバイナリ群のクラスタリング処理が施され代表RTバイナリが選出される．

リバーストレーサ[4]とはプログラム実行履歴(トレース)から，それを再現する実行バイナリを生成する技術である．今回我々が対象としているCPU2000プログラム群はユーザ実行時間が支配的であるため，複数コンテキスト実行や非同期割り込みに注意を払う必要がない．このためRT-liteは[4]で提案されているものより簡素なものとなっている．

これらの代表RTバイナリはRTLレベル論理シミュレーションに適用されたり，また再度トレースを採取した後トレースドリブンシミュレーションに適用されるなどして評価される．

2.2 RTバイナリの形式

GREPTの出力であるRTバイナリの形式について述べる．RTバイナリは図2のようなELF形式の実行バイナリ形式になっている．

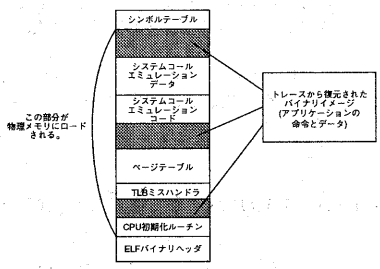

図2 RTバイナリの形式

オペレーティングシステム(以下OS)未開発時の論理シミュレーションへの適用を考慮し，RTバイナリはOSエミュレーションのための最小限のコードおよびデータを保持している．これにより性能評価に必要なメモリ量や評価環境の制約を緩和することができる．OSエミュレーションルーチンには以下のものが含まれる．

- CPU初期化ルーチン
- トラップハンドラ(TLBミスハンドラ，ページテーブル，レジスタウィンドウspill/fillハンドラ)
- システムコールエミュレーション用領域(命令コード+データ)

RTバイナリはシステムコール後のレジスタやメモリの内容をデータとして保持し，システムコールの代わりにメモリ転送を行うことでシステムコールをエミュレートする．

2.3 トレースサンプラ

RTバイナリ生成に必要な情報をトレース情報として収集するため，Sun Microsystemsによって開発された命令トレーサshade v6[7]を元にトレースサンプラを作成した．トレースサンプラはランダムな命令間隔でサンプリングトレースを出力する．

RTバイナリ生成には実行される命令語およびアクセスされるデータが必要である．またシステムコールエミュレーションのためにシステムコール情報も必要である．これらの要求を満たすためトレースサンプラは以下の情報を収集する．

初期状態情報 サンプリング開始時のレジスタ値，スタック領域データ．

☆ REpresentative Performance Test Selector

実行履歴情報　PC, 命令語，アクセスアドレス，メモリからロードされたデータ．1命令毎に収集．

システムコール情報　呼び出し元PC，システムコールにより変更されたレジスタ・メモリ内容．システムコール毎に収集．

2.4 RT-liteによるワークロード生成

RT-liteは大きく3つの処理フェーズに分かれている．

- 仮想メモリイメージ復元
- OSエミュレーションルーチン作成
- バイナリ出力

2.4.1 仮想メモリイメージ復元

RTバイナリの仮想アドレス上のメモリイメージを仮想メモリイメージと呼ぶこととする．RT-liteの仮想メモリイメージ復元方式の概要を図3に示す．ま

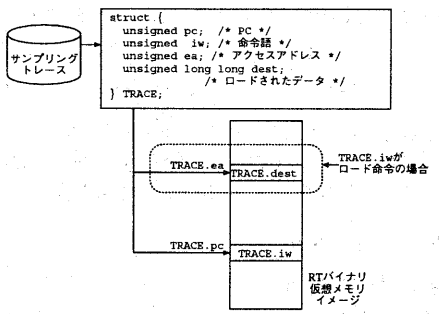

図3　RT-liteの仮想メモリイメージ復元

ずサンプリングトレースファイルに記録されているスタック領域データを復元する．その後実行履歴情報を読み出しながら以下の操作を行う．

(1)　PCが指し示しているアドレスの内容として命令語をセットする．

(2)　もし命令語がロード命令ならば，アクセスアドレスの内容としてロードされたデータを格納する．

2.4.2 OSエミュレーションルーチン作成

仮想メモリイメージ復元で使用されなかった仮想アドレスにOSエミュレーションルーチンを作成する (図4)．

CPU初期化ルーチン作成

RTバイナリ動作開始直後に走行するルーチンの作成を行う．このルーチンでは特権モードでしかアクセスできないレジスタの初期化，TLBの初期化，トラップハンドラの初期化が行われる．

トラップハンドラ作成

TLBミスハンドラ，レジスタウィンドウspill/fillハンドラの作成を行う．さらにRTバイナリの仮想メモリイメージからに物理メモリバイナリイメージを作りページテーブルを作成する．

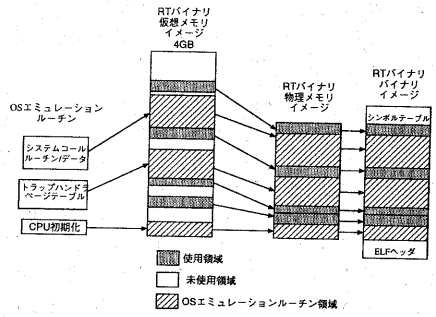

図4　RTバイナリの仮想・物理メモリイメージとバイナリイメージ

システムコールエミュレーション用領域作成

システムコールエミュレーションルーチンおよびシステムコールによって変更されたメモリ内容をバイナリの一部として格納する．同時にシステムコール呼び出し部分をシステムコールエミュレーションルーチンへのジャンプ命令へ書き換える．

2.4.3 バイナリ出力

OSエミュレーションルーチン作成によって生成された物理メモリバイナリイメージに，ELFヘッダとOSエミュレーションルーチンのシンボル情報を付加しRTバイナリとして出力する．シンボル情報は主にOSエミュレーションルーチンのデバッグのために使用される．

2.5 REPTSによる代表RTバイナリ選出

REPTSは，RT-liteによって生成されたRTバイナリ群からクラスタリングを用いて代表となるRTバイナリを選出する．

選出指標として時間的に低コストに算出できる指標が望ましい．RTバイナリは通常の実行バイナリにはない以下の特徴を持つ．

- RTバイナリにはトレースに含まれている (実行された) 命令しか存在しない．すなわち命令フローが異なるバイナリは，異なった命令コードをイメージを持つ．
- 初期データも全てRTバイナリバイナリに含まれている．
- RTバイナリサイズそのものがメモリのフットプリントになっている．

我々はこれらの特性に着目しRTバイナリの実行バイナリイメージそのものが特性を表しているのではないかと推測した．そこで今回はRTバイナリのバイナリ比較を用いて代表選出を行うことにした．

以下，クラスタリングに使用されるRTバイナリ間の距離の定義，クラスタ内の代表RTバイナリの定義，クラスタ間の距離の定義，および選出手順の概要を説明する．

2.5.1 RTバイナリ間の距離の定義

ある2つのRTバイナリb_iとb_jの距離$l(b_i, b_j)$を

以下のように定義する．ここで $f(b_i)$ は b_iのファイルサイズ，$d(b_i, b_j)$ を b_i と b_jのイメージ比較による相違バイト数であるとする．

$$l(b_i, b_j) = \frac{d(b_i, b_j)}{max(f(b_i), f(b_j))} \quad (1)$$

$l(i, j)$ は 0 以上 1 以下の値をとる．値が小さいほど類似度が高いと判断する．

2.5.2 クラスタ内の代表RT バイナリの定義

クラスタ C のRT バイナリそれぞれに対して以下の評価値を計算し，式 (2) で計算される評価値 $r(b_i)$ がもっとも小さいものを代表RT バイナリであると定義する．

$$r(b_i) = \sum_{b_j \in C} l(b_i, b_j) \quad (2)$$

さらにクラスタ間の距離は各クラスタの代表RT バイナリ間の距離であると定義する．

2.5.3 選出手順

(1) 式 (1) に基づいて類似度行列を作成する．

(2) 凝集的アルゴリズム[5]を用いてデンドログラムを形成する．凝集的アルゴリズムの最中に，新規に生成されたクラスタについては再度代表RT バイナリの選出を行う．

(3) 所望のクラスタ数に分割した後，各クラスタの代表RT バイナリを最終的なものとする．ただしクラスタ内のRT バイナリ数が 1 であった場合には，そのクラスタを破棄しもっとも要素数の大きいクラスタを分割する．

クラスタ C_i内のRT バイナリ数をクラスタの重み w_i と定義する．要素数が少ないクラスタはクラスタの重みが小さいため性能評価上無視できる．このため要素数が 1 のクラスタを破棄する．

2.6 代表RT バイナリを用いた CPI 推定

代表RT バイナリを用いた評価では，クラスタの重みによって重みづけ評価される．CPI 推定値 (CPI') の算出方法は以下の通りである．

$$CPI' = \frac{\sum_{C_{all}} CPI_{R_i} \times w_i}{\sum_{C_{all}} w_i} \quad (3)$$

ここで C_{all}は最終的に得られたクラスタ集合，CPI_{R_i} はクラスタ C_iの代表RT バイナリの CPI である．

2.7 クラスタリング方式の評価指標

式 (1) で示したように実行バイナリのバイナリイメージ比較によるクラスタリングと代表RT バイナリ選択を行っているため，CPI の観点から適切であるかどうか検証する必要がある．そこでクラスタリングの評価指標として，「平均クラスタ内 CPI 変動係数」に基づく「クラスタ度」と「平均代表度」を導入し検証する．

2.7.1 平均クラスタ内 CPI 変動係数 ($COV_{cluster}$)

平均クラスタ内 CPI 変動係数 $COV_{cluster}$を次式によって定義する．

$$COV_{cluster} = \frac{\sum_{C_{all}} COV_{C_i} \times w_i}{\sum_{C_{all}} w_i} \quad (4)$$

ここで COV_{C_i}はクラスタ内 CPI の変動係数である．

2.7.2 クラスタ度 (Q_{CPI})

CPI に関するクラスタリングの質 Q_{CPI}を，$COV_{cluster}$と全RT バイナリの CPI の変動係数COV_{all}との比で定義する．

$$Q_{CPI} = \frac{COV_{cluster}}{COV_{all}} \quad (5)$$

Q_{CPI} の値が小さいほどうまくクラスタリングできていると判断できる．また Q_{CPI}が 1 を超えた場合クラスタ内の CPI のばらつきが全体のばらつきより大きいことを意味している．この場合クラスタリングの効果が全くない，あるいは逆効果であったと判断できる．

2.7.3 平均代表度 ($REPR$)

平均代表度 $REPR$ を次式によって定義する．

$$REPR = \frac{\sum_{C_{all}} \frac{|CPI_{R_i} - CPI_{C_i}| \times w_i}{s(C_i)}}{\sum_{C_{all}} w_i} \quad (6)$$

ここで CPI_{C_i}はクラスタ C_iに含まれるRT バイナリの平均 CPI，$s(C_i)$ は CPI の標準偏差である．この値が小さいほど代表RT バイナリとして質が高い．

3. 実験結果

3.1 評価環境

スーパスカラシミュレータ**paratool**[10]を用いて CPI 測定を行った．測定にあたって SPARC64-III プロセッサ[11]相当のアーキテクチャパラメータを使用した (表 1)．

表 1 対象アーキテクチャパラメータ

fetch 幅/issue 幅	4/4
機能ユニット	IALU:2,LD/ST:2 FADD:1,FMUL:1
命令 1 次キャッシュ	64K ダイレクトマップ
データ 1 次キャッシュ	64K ダイレクトマップ
2 次キャッシュ	8M ダイレクトマップ HIT 時 8cycle/MISS 時 100cycle

3.2 対象ベンチマーク

CPU2000 ベンチマークテストから以下の 8 本のプログラム (整数系プログラム群 CINT2000 より 4 本，数値計算系プログラム群 CFP2000 より 4 本) について，RT バイナリによる CPI 推定に関する実験を行った．

CINT2000

`176.gcc, 252.eon, 254.gap, 255.vortex`

CFP2000

`173.applu, 177.mesa, 188.ammp, 301.apsi`

表 2 に全RT バイナリの CPI の変動係数をしめす．この値が大きいプログラムは実行中に大きく CPI が変

動することを意味している．

表 2　全RTバイナリの変動係数

176.gcc	0.508	252.eon	0.040
254.gap	0.183	255.vortex	0.222
173.applu	0.133	177.mesa	0.157
188.ammp	0.175	301.apsi	0.525

3.3 RTバイナリ作成

各プログラム毎に 60 個のサンプリングトレースを採取しRTバイナリの生成を行った．1 つのサンプリングトレースは 100M 命令を含む．クラスタリングにより 60 個のRTバイナリから 5 つの代表を選択し評価を行った．

3.4 CPI 推定値誤差

代表RTバイナリの CPI を測定し，式 (3) を用いて CPI 値を推定した．ここでは全RTバイナリの平均 CPI を正しい CPI 値と仮定し，推定値の誤差を計算する．平均 CPI と CPI 推定値を図 5 に，推定値誤差を表 3 に示す．176.gcc を除けば CPI 推定値の誤差は 10% 以内であった．(平均で 7．7% の誤差)

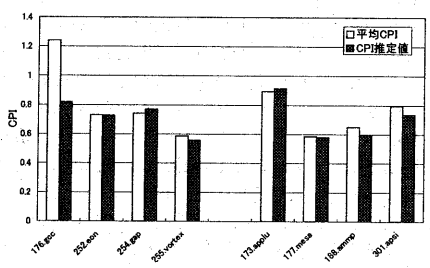

図 5　バイナリ比較による CPI 推定値

表 3　CPI 推定値の誤差

176.gcc	-33.8%	252.eon	-0.3%
254.gap	+3.9%	255.vortex	-4.9%
173.applu	+2.1%	177.mesa	-1.0%
188.ammp	-8.4%	301.apsi	-7.5%

3.5 クラスタ度 (Q_{CPI}) による評価

バイナリイメージの差によるクラスタリングの質を評価するために，Q_{CPI}を算出した．表 4 に各プログラムの Q_{CPI} を示す．255.vortex を除いて Q_{CPI} が 1.0 より小さい．とくに CFP2000 より選択したプログラム (173.applu, 177.mesa, 301.apsi) に関しては，CINT2000 よりクラスタリングの質が高いと言える．しかしながら176.gcc, 254.gap, 188.ammp では 0.82〜0.91 と，全RTバイナリのばらつきとほぼ同じになった．式 (1) によるクラスタリングはあまり効率の良いものではないことがわかる．

表 4　クラスタリングの質 Q_{CPI}

176.gcc	0.90	252.eon	0.61
254.gap	0.91	255.vortex	1.03
173.applu	0.14	177.mesa	0.00
188.ammp	0.82	301.apsi	0.41

3.6 平均代表度 ($REPR$) による評価

代表RTバイナリ選択の代表度 $REPR$ を表 5 に示す．$REPR$ は代表RTバイナリの CPI 誤差を標準化したものとなっている．この表 5 からわかるように，CFP2000 より選出したプログラムの $REPR$ の方が良好な結果を示している．

表 5　平均代表度 $REPR$

176.gcc	0.66	252.eon	0.76
254.gap	0.50	255.vortex	0.44
173.applu	0.25	177.mesa	0.39
188.ammp	0.42	301.apsi	0.46

4. 考　　察

4.1 クラスタリング可能性について

対象プログラムのクラスタリング可能性を示しておく．式 (1) の代りにRTバイナリ間の CPI の差を用いてクラスタリングした結果を図 6 に示す．また表に Q_{CPI}を示す．

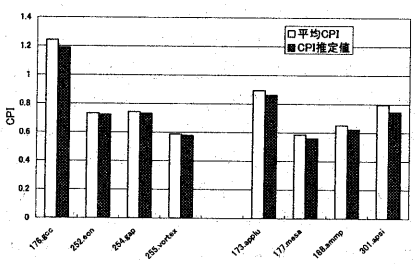

図 6　CPI を距離に使ったクラスタリング

表 6　CPI を距離に用いたクラスタリングの Q_{CPI}

176.gcc	0.03	252.eon	0.40
254.gap	0.02	255.vortex	0.07
173.applu	0.12	177.mesa	0.03
188.ammp	0.29	301.apsi	0.03

これらの結果からわかるように，対象プログラム自体は CPI に関してクラスタリングが可能であることがわかる．ところが全RTバイナリに対し CPI を算出すると，サンプリング数を増やしたときに CPI 算出

時間が問題となる．このためCPIを距離に使うことは現実的ではない．

4.2 RTバイナリ間の距離について

RTバイナリ間の距離を式(1)で定義した．CPI推定値誤差は少ないものの，クラスタリングの質，代表RTバイナリ選出の点で問題があった．式(1)の距離定義には以下のような問題がある．

- データの内容の違いが距離に影響を与える．
- 命令語の相違とデータの相違が同じ指標や重み付けで評価されている．

性能への影響を考えた場合，データの内容は問題ではなくアクセスしたメモリアドレスが問題になる．またCPU2000ベンチマークテストのように単一プロセスを対象に評価する場合，命令アドレスによって命令語が一意に決まることが多い．これらからRTバイナリ間の距離をRTバイナリのメモリフットプリントの相違にする方が適正であると考えられる．

また命令アドレスが異なる場合プログラムの処理内容が違うことを意味する．一方データに関してはアドレスの絶対値ではなくアクセスアドレスの分布が性能に影響する．メモリフットプリントの相違を指標にするとしても，命令とデータとで異なる指標や重み付けを行う方が良いと考えられる．

4.3 圧縮形式としてのRTバイナリ

RTバイナリをトレースファイルの圧縮表現として捉えることも可能である．最近のハードウェアの物量(とくにキャッシュ)の増加に伴い，性能評価に必要なトレースデータサイズが増加する傾向にある．そこでトレースファイルを実行可能なバイナリとして保存し，必要に応じて実行トレースを生成するという手法をとることでディスク使用量を大きく削減できる．たとえば100Mステップの実行トレース(2Gバイト)が，`176.gcc`で10Mバイト，`255.vortex`で32Mバイト程度のRTバイナリに変換することができている．

またRTバイナリは実行時スナップショット(メモリイメージ)の圧縮表現にもなっている．トレースデータに関わりのある命令コードとデータしか保持しないためサイズを小さく押さえることができる．

5. まとめ

我々は一般アプリケーションにも適用できる性能テストプログラム作成を行うシステム***GREPT***を作成した．これはトレースサンプリング技術・リバーストレーサ技術・クラスタリング技術によって代表RTバイナリを選出し，評価コストを大幅に減らすことを特長としている．

本稿では低コストな代表RTバイナリ選出を実現するため，RTバイナリのバイナリイメージの違いを距離と定義し凝集的アルゴリズムを用いてクラスタリングを行った．クラスタリングの質としては改善の余地はあるが，代表バイナリから推定したCPI誤差は平均で7.7% (`176.gcc`を除いて8.4%以下)となった．

今後，評価対象プログラム(ベンチマークテスト)が時間的・空間的に巨大化していく傾向にある．このため高速に有用な縮小テストプログラム生成を行う技術が重要となっていく．時間的に低コストで算出でき，かつ有効な「距離」について検討していく予定である．

また今回検討しなかったが，本手法によって得られるCPI推定値の信頼度指標についても検討を行っていく予定である．

参考文献

1) S. Laha, J. Patel, R. Iyer, "Accurate low-cost methods for performance evaluation of cache memory systems", IEEE Trans. Comput. 37,11, 1325-1336, 1988.
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 Stream", Workshop on Workload Characterization (WWC 2000), Sep 2000.
4) M. Sakamoto, L. Brisson, A, Katsuno, A, Inoue, Y. Kimura, "Reverse Tracer: A Software Tool for Generating Realistic Performance Test Programs", HPCA, Feb 2002.
5) M.J.A. Berry and G. Linoff(SASインスティチュートジャパン, 江原, 佐藤訳), "データマイニング手法: 営業，マーケティング，カスタマーサポートのための顧客分析", 海文堂, Sep 1999.
6) J.L.Henning, SPEC CPU2000: Measuring CPU Performance in the New Millennium, IEEE Computer, 33(7):28-35, Jul 2000. SPEC CPU2000 Press Release FAQ, available at http://www.spec.org/osg/cpu2000/press
7) "Shade and Spixtools", Available at `http://www.sun.com/microelectronics/shade/`
8) 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", システム評価研究会, Feb 2002.
9) 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", システム評価研究会, Feb 2002.
10) 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
11) "SPARC64-III User's Guide", HAL Computer Systems, Inc, May 1998.