計算機アーキテクチャ　144−31
（２００１．　７．２７）

計算機システムにおける分岐予測外れパスの影響

大河原　英喜[†]　河　場　基　行[†]　安　里　　彰[†]

計算機システムの性能評価に用いられるシミュレータには、実行ドリブン型シミュレータ (EDS) とトレースドリブン型シミュレータ (TDS) の大きく 2 種類がある。TDS を用いた性能評価は、動的な振る舞いが評価できず、評価誤差が生じるという問題点が知られている。本論文では、TDS において分岐予測外れパスが評価されないことに着目し、分岐予測外れパスが計算機システムに如何に影響を与えるかについて調査した。評価の結果、極端に違いが現れる例では、分岐予測外れパスの命令をシミュレーションするか否かで、実行サイクル数に約 93% の違いが見られた。また、SPEC INT92 で評価を行った結果、fetch 命令数で 30%、実行サイクル数では 3% 程度の違いが見られた。

The influence of branch miss-predicted path to computer system performance

Hideki Okawara,[†] Motoyuki Kawaba[†] and Akira Asato [†]

To evaluate computer system performance, two types of simulators, Execution Driven Simulator(EDS) and Trace Driven Simulator(TDS), are commonly used. The drawback of TDS is its inability of simulating dynamic behavior, for example, speculative instructions fetching. In this paper, we focus on speculative instruction fetching and investigate the influence of branch miss-predicted path to the computer system performance. In a extreme case, the evaluation error of TDS is about 93% in execution time. In SPEC INT92 benchmark,the evaluation errors are 30% in the number of instructions fetched and 3% in the execution time.

1. はじめに

プロセッサのマイクロアーキテクチャやメモリアーキテクチャなど, 計算機システムの性能評価にはソフトウェアシミュレータが用いられることが多い。ソフトウェアシミュレータには、シミュレーション方式において, 大きく下に示す 2 種類がある。

実行ドリブン型シミュレータ (EDS)[1)2)]　性能評価時に命令を動的にエミュレーションし、それと連動してシミュレーションを行う方式。動的に振る舞いが変化する場合にも正しく評価することができる。

トレースドリブン型シミュレータ (TDS)　予め採取された、ある環境下における実行時の命令列のトレースデータを基にシミュレーションを行う方式。動的に振る舞いが変化する場合は評価できない。

TDS で正しく評価できない振る舞いの具体的な例として、以下の様なものが挙げられ、これらを解決する TDS 手法の研究も行われている[3)]。

- マルチプロセッサシステムにおけるロック変数獲得の様な、処理タイミングによって命令フローが変化するアプリケーションの評価ができない。
- 分岐予測ミス時に誤って投機実行される、分岐予測外れパスがシミュレーションに含まれず、評価誤差につながる。

分岐予測外れパスがシミュレーションに含まれないという問題点はよく知られているが、その影響に関する定量的な評価はあまりされていない。単体プロセッサにおいて分岐予測外れパスの影響を評価した研究[4)] では、分岐予測外れパスにおけるキャッシュミスがプリフェッチ効果を持つことを示し、 EDS と TDS とでキャッシュの振る舞いに違いがあることを示した。この評価では、分岐予測外れパスが IPC に与える影響は小さいという結論が得られた。

しかし、分岐予測外れパスによってキャッシュミスが頻発する場合や、メモリアクセスが性能のボトルネックとなるマルチプロセッサシステムでは、分岐予測外れパスの影響はより大きいと考えられる。そこで、本論文では単体プロセッサに加えマルチプロセッサシステムにおける分岐予測外れパスの影響に関する評価を行う。

2. 分岐予測ミス時の動作について

本章では、分岐予測ミス時の EDS と TDS の動作の違いについて、一例をとりあげて述べる。

分岐予測機構を備えたプロセッサにおいては、分岐予測ミス時にも投機実行が行われる。 EDS では分岐予測外れパスもシミュレーションされるが、 TDS では分岐

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

予測外れパスはシミュレーションされず、評価誤差につながる。分岐予測ミス時の実機、EDS でのパイプラインの動作例を図 1に, TDS での動作例を図 2に示す。これらの例では、命令 0 の分岐命令が分岐予測ミスを起こすとする。

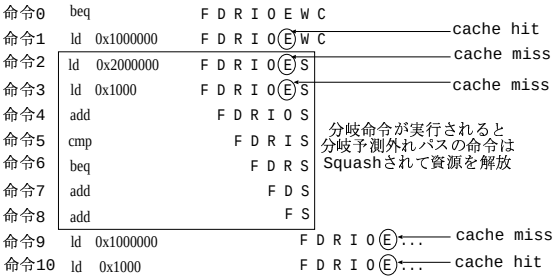

図 1 分岐予測ミス時の動作例 (実機、EDS)

実機や EDS では、分岐予測外れパスの命令 (命令 2 〜命令 8) が投機的に fetch されパイプラインに投入される。分岐命令（命令 0）が実行され、正しい分岐方向が確定したら、分岐予測外れパスの命令は squash され、Reservation Station(RS) や renaming register など、確保していた資源の解放を行う。但し、一度、出されたメモリアクセス要求に関しては処理が続行される。

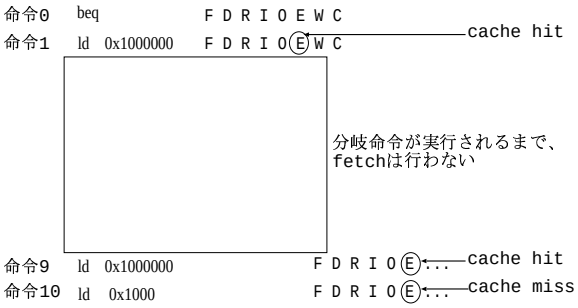

図 2 分岐予測ミス時の動作例 (TDS)

それに対し、TDS では分岐外れパスの命令がトレースデータに含まれないため、分岐命令（命令 0）が実行されて正しい分岐方向が確定するまで、シミュレーションは行われない。

この様に、EDS と TDS とには、分岐予測外れパスの命令が実行されるか否かという違いがあり、EDS の方がより正確なシミュレーションを行っている。この、分岐予測外れパスが実行されるか否かの違いが計算機システムに与えうる影響として以下の点が挙げられる。

キャッシュ状態の変化 分岐予測外れパス中にあるメモリアクセス命令がシミュレーションされるか否かでキャッシュ状態に違いが現れる。図 1、図 2の例では、以下の 2 通りのケースを示している。

- 命令 1 の時点ではアドレス 0x1000000 のデータはキャッシュ上にあるとする。EDS では、命令 2 が実行されるとアドレス 0x1000000 とアドレス 0x2000000 が干渉し、アドレス 0x1000000 のデータはキャッシュ上から追い出される。そのため、命令 9 ではアドレス 0x1000000 はキャッシュミスとなる。一方、TDS では命令 9 においてアドレス 0x1000000 はキャッシュヒットとなる。
- EDS では、命令 3が実行されるとキャッシュミスを起こしてアドレス 0x1000 のデータを読み込む。そのため、命令 10 ではアドレス 0x1000 はキャッシュヒットとなる。一方、TDS では命令 10 がアドレス 0x1000 への初アクセスであり、この時点でキャッシュミスを起こす。この様に、分岐予測外れパスにおけるメモリアクセスがプリフェッチ効果を持つ場合がある。

分岐予測外れパスによる資源の枯渇 EDS では、パイプラインに投入された分岐予測外れパスの命令がハードウェア資源の確保を行い、squash されて資源を解放するまでの間、資源が枯渇する可能性がある。それに対し、TDS では分岐予測外れパスの命令はパイプラインに投入されず、分岐予測外れパスの命令による資源の枯渇は起きない。

近年の計算機システムの動向を考えると、プロセッサ内部の資源の枯渇よりも、メモリアクセスがより性能に影響を与えると考えられる。そこで本論文では、特に、分岐外れパスによるキャッシュ状態の変化に着目し、分岐予測外れパスが性能に与える影響について評価する。

3. 分岐予測外れパスの影響に関する評価

3.1 評価環境

3.1.1 *SEEDS(SPARC Emulateor and Execution Driven Simulator)*

本節では、我々が開発した実行ドリブン型シミュレータ, *SEEDS* の概要を述べる。本論文における性能評価はこの *SEEDS* を用いて行った。

3.1.1.1 *SEEDS* の構成

SEEDS は、*SPARC V9* 命令セットに対応した実行ドリブン型シミュレータで、マルチプロセッサシステム、マルチスレッドアプリケーションのシミュレーションを行うことができる。*SEEDS* の構成を図 3に示す。

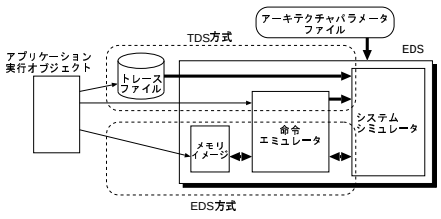

図 3 *SEEDS* の構成

*SEEDS*は、*SPARC V9*対応の命令エミュレータ部、サイクルベースでパイプライン動作やキャッシュ動作などをシミュレーションするシステムシミュレータ部から構成される。*SEEDS*では、入力ファイルや分岐予測ミス時の挙動に関して、以下の３通りの方式のシミュレーションを行うことができる。

EDS 方式　実行オブジェクトを入力とし、動的に命令をエミュレーションしながらシミュレーションを行う。分岐予測外れパスのシミュレーションも行い、分岐方向が確定されたら分岐命令時の正しい状態を復帰し、正しいパスの命令から再開する。

TDS-e 方式　実行オブジェクトを入力とし、動的に命令をエミュレーションする。分岐予測外れパスのシミュレーションは行わず、分岐方向が確定するまで命令 *fetch* は行わない。

TDS-t 方式　従来の *TDS* と同様にトレースファイルを入力とし、分岐予測外れパスのシミュレーションは行わない。

*3.2*節以降では、*EDS* 方式と *TDS-e* 方式のシミュレーション結果の比較を行うことで、計算機システムにおける分岐予測外れパスの影響を評価した。どちらのシミュレーション方式においても、実行オブジェクトを基に動的に命令をエミュレーションしているため、両者の違いは、分岐予測外れパスのシミュレーションを行うか否かという点だけである。以後、*TDS-e* 方式のことを単に *TDS* 方式と記述する。

3.1.1.2　システムシミュレータ部の概要

システムシミュレータ部の概要を図 *4*に示す。システムシミュレータ部は、パイプライン動作をシミュレーションする *CPU* モデルや、キャッシュ動作、メモリアクセス動作をシミュレーションするメモリモデルから構成されている。

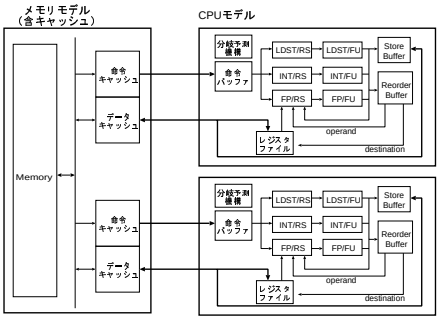

図 4　システムシミュレータ部の概要

CPU モデルの主な特徴は以下の点である。

- *8* 段パイプラインのスーパスカラプロセッサ
- *out-of-order issue,in-order commit*
- 動的分岐予測機構 *(gshare)*

メモリモデルの主な特徴は以下の点である。

- マルチプロセッサに対応
- バスアーキテクチャ、キャッシュプロトコル *(MOESI*プロトコル*)* を考慮したシミュレーション
- キャッシュは３階層まで

3.2　サンプルプログラムによる評価

本節では、分岐予測外れパスの影響が大きく現れるサンプルプログラムを作成し、最悪なケースでの分岐予測外れパスの影響を調べる。

3.2.1　サンプルプログラムの概要

分岐予測外れパスの影響が大きく現れる場合として、以下の様な場合が考えられる。

- 分岐予測ミス率が高く、分岐予測外れパスの命令が多く実行される場合
- 分岐予測外れパスでのキャッシュアクセスによりキャッシュミス数が増加する場合

これらのことを考慮し、予測外れパスでキャッシュミスが頻発する例として、以下に示すサンプルプログラムを用いて評価を行った。

```
int P[4] = {1, 0, 1, 0};
int sum,array[STRIDE/sizeof(int)+1];

main() {
  int i;
  for (i = 0; i < 10000; i++) {
    if (P[i%4]) {
      sum += *(array + STRIDE*(1-P[i%4]));  /* 処理1 */
    } else {
      sum += *(array + STRIDE*P[i%4]);      /* 処理2 */
    }
  }
}
```

このサンプルプログラムでは、表 *1*に示す様に、*if* 文での分岐予測がヒットすれば **(array)* を、ミスすれば **(array+STRIDE)* をアクセスする。*STRIDE* がキャッシュサイズと等しい場合には、**(array)* と **(array+STRIDE)* は同一キャッシュラインとなる。そのため、連想度が *direct-mapped* であると仮定すれば、分岐予測外れパスが実行されると以下の様に２回のキャッシュミスが発生すると考えられる。

(1)　分岐予測外れパスにおいて **(array+STRIDE)* がキャッシュミスを起こし、**(array)* をキャッシュ上から追い出す。

(2)　その後、正しいパスに復帰した際に、**(array)* へのアクセスがキャッシュミスを起こす。

表 1　サンプルプログラムのメモリアクセスパターン)

P[i%4]	分岐予測	処理	アクセスアドレス
1	ヒット	処理１	*(array + STRIDE*(1-1))
1	ミス	処理２	*(array + STRIDE*1)
0	ヒット	処理２	*(array + STRIDE*0)
0	ミス	処理１	*(array + STRIDE*(1-0))

3.2.2　評価条件

*3.2.1*に示したサンプルプログラムを用いて、最悪なケースでの分岐予測外れパスの影響を評価した。評価

は、*UP*(単一プロセッサ)、*MP*(マルチプロセッサ) について行った。*MP* とは、*4CPU* 上でサンプルプログラムを 1 プロセスづつ、計 4 プロセスを同時に実行した場合である。評価に用いた *CPU* の主なパラメータを表 *2*に示す。

表 2 UP、MP の評価パラメータ (サンプルプログラム)

fetch 幅	8 命令 /cycle
演算器 (int,fp,ldst,br)	(4,4,4,1)
RS size(int,fp,ldst,br)	(32,16,24,16)
レジスタ (int,fp,cc,y)	(32,32,32,1)
1 次キャッシュ	I1 + D1(各 64kB,direct) レーテンシ 17 cycle
外部キャッシュ	unify(2MB,direct) レーテンシ 100 cycle

ここではキャッシュは 2 階層とし、**(arrary)* と **(array+STRIDE)* がキャッシュ干渉を起こしやすい様、1 次キャッシュ、外部キャッシュの連想度を *direct mapped* とした。サンプルプログラムの *STRIDE* の値を以下の 3 通りに変化させて評価を行った。

32kB **(array)* と **(array+STRIDE)* はキャッシュ干渉を起こさない

64kB **(array)* と **(array+STRIDE)* は 1 次キャッシュで干渉を起こす

2MB **(array)* と **(array+STRIDE)* は 1 次キャッシュ、外部キャッシュで干渉を起こす

3.2.3 性能評価 (サンプルプログラム)

サンプルプログラムにおける *EDS* 方式と *TDS* 方式のシミュレーション結果の比較を、*UP* の場合を図 *5*、*MP* の場合を図 *6*に示す。図中には、*EDS* 方式のシミュレーション結果を基準 *(=1)* とした場合の、*TDS* 方式のシミュレーション結果の相対値を示している。

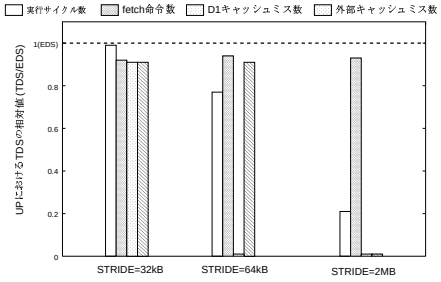

図 5 サンプルプログラムにおける TDS と EDS の違い (UP)

3 種類の *STRIDE* に関して比較すると、いずれの場合にも *fetch* 命令数は約 *7%* の差が見られる。キャッシュミス数は、*STRIDE* が *64kB* の場合には 1 次キャッシュ、*STRIDE* が *2MB* の場合には 1 次キャッシュと外部キャッシュにおいて、*EDS* 方式と *TDS* 方式に非常に大きな違いが見られる。実行サイクル数に関しては、*STRIDE* が *32kB* の場合には *EDS* 方式と *TDS* 方式に

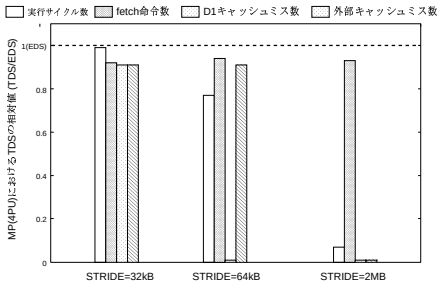

図 6 サンプルプログラムにおける TDS と EDS の違い (MP)

はほとんど違いがない。しかし、*STRIDE* が *64kB* の場合には約 *23%*, *STRIDE* が *2MB* の場合には約 *79%* もの違いが見られる。

MP の場合にも、*UP* の場合とほぼ同じ結果が得られたが、*STRIDE* が *2MB* の場合の実行サイクル数において、*MP* の方が *EDS* 方式と *TDS* 方式の違いが大きく、約 *93%* の違いが見られた。

3.3 ベンチマークを用いた性能評価

3.3.1 評 価 条 件

本節では、分岐予測外れパスの影響の、より一般的な評価として、ベンチマークを用いた評価を行った。ベンチマークとしては、シングルスレッドアプリケーションとして、分岐予測ミス率が大きい *olden(mst,health)*[5)6)]*,SPEC INT92(espresso.gcc,li)* と、キャッシュミス率が大きい *CPU2000(art)* を用いた。又、マルチスレッドアプリケーションとして、*SPLASH2(fft,lu,radix,cholesky)*[7)] も評価に用いた。

主な評価パラメータを表 *3*に示す。*3.2*節の評価ではキャッシュミスを頻発させるためにキャッシュの連想度を *direct-mapped* としたが、ここではより一般的なパラメータとして、1 次キャッシュの連想度を *4way*、外部キャッシュの連想度を *2way* とした。

表 3 UP、MP の評価パラメータ (ベンチマーク)

CPU 数	1 CPU
CPU fetch 幅	8 命令 /cycle
演算器 (int,fp,ldst,br)	(4,4,4,1)
RS size(int,fp,ldst,br)	(32,16,24,16)
レジスタ (int,fp,cc,y)	(32,32,32,1)
1 次キャッシュ	I1 + D1(各 64kB,4way) レーテンシ 17 cycle
外部キャッシュ	unify(2MB,2way) レーテンシ 100 cycle

*3.2*節と同様に、*UP* と *MP(4CPU)* の 2 種類の計算機システムに対して評価を行った。*MP* モデルの評価に関しては、シングルスレッドアプリケーションでは同一ベンチマークを各 *CPU* 上で 1 プロセスづつ、計 4 プロセスを同時実行した。マルチスレッドアプリケーションである *SPLASH2* では、主要部を 4 スレッドに分割し、*4CPU* 上で 1 スレッドづつ処理を行った。

3.3.2 評価結果 (SPEC,olden)

olden,SPEC INT92,art を用いたシミュレーションにおける、*EDS* 方式と *TDS* 方式のシミュレーション結果の比較を、図 *7(UP)*、図 *8(MP)* に示す。

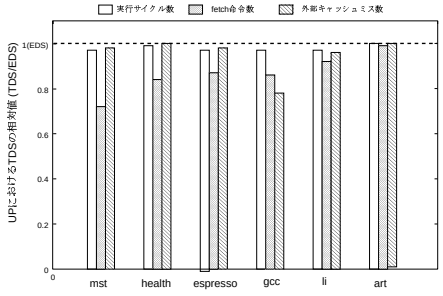

図 7 SPEC,olden における EDS と TDS の違い（UP）

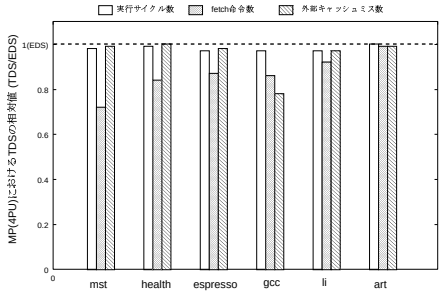

図 8 SPEC,olden における EDS と TDS の違い（MP）

図 *7*、図 *8*を比較すると、*UP,MP* の結果に大きな違いは見られなかった。各ベンチマークの結果を比較すると、*fetch* 命令数に関しては、最大で *30%* 近くの違いが得られ、*olden(mst,health)* に最も大きな違いが見られた。しかし、実行サイクル数に関してはいずれのベンチマークでも大きな違いは見られず、*EDS* 方式と *TDS* 方式の実行サイクル数の違いは最大でも約 *3%* であった。外部キャッシュミス数の違いは *gcc* が最も大きく、約 *20%* の違いが見られる。他のベンチマークに関しては、外部キャッシュミス数の違いは小さく、*5%* 未満の違いしか見られなかった。

3.3.3 評価結果 (SPLASH2)

fft,lu,radix,cholesky の 4 種類について評価を行った結果、*EDS* 方式と *TDS* 方式の実行サイクル数の違いは最大 *1.5%* であった。また、*fetch* 命令数、外部キャッシュミス数の違いも小さく、*fetch* 命令数で *3.5%*、外部キャッシュミス数で *3%* の違いしか見られなかった。

4. 考　　察

4.1 サンプルプログラムによる性能評価

サンプルプログラムによる評価 *(*図 *5*、図 *6)* では、*STRIDE* が *64kB,2MB* の場合に *EDS* 方式と *TDS* 方式の実行サイクル数に違いが見られた。これは、キャッシュミス数の違いが原因として考えられる。ここで、プログラムの性質を明らかにするために、*EDS* 方式,*TDS* 方式における分岐予測ミス率、1 次キャッシュミス率、外部キャッシュミス率を、図 *9*に示す。各ミス率は、*(*ミス数*)/(*全 *commit* 命令数*)* として求めた値である。

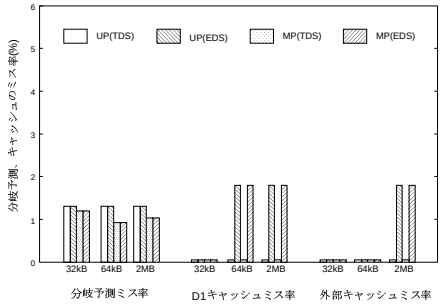

図 9 サンプルプログラムの性質

分岐予測ミス率は、いずれの場合にも *1%* 前後である。キャッシュミス率を見ると、*STRIDE=32kB* の場合には非常に小さいが、*STRIDE* が *64kB* の場合の 1 次キャッシュ、*STRIDE* が *2MB* の場合の 1 次キャッシュと外部キャッシュは、*2%* 近くのキャッシュミス率となっている。この、キャッシュミス率にして *2%* 近くの違いが実行サイクル数に大きく影響を与えている。

また、*STRIDE* が *2MB* の場合には、外部キャッシュミスによる、メモリやバスに対するアクセスの増加が実行サイクル数に影響を与えている。この様な場合、マルチプロセッサシステムでは単一プロセッサより分岐予測外れパスの影響がより顕著に現れている。

4.2 ベンチマークによる性能評価

ベンチマークによる評価 *(*図 *7,* 図 *8)* において、*fetch* 命令数や外部キャッシュミス数の *EDS* 方式と *TDS* 方式の違いが、実行サイクル数にそれほど影響を与えない原因として、以下の様な点が考えられる。

- 分岐予測ミス率が小さく、分岐予測外れパスとして実行される命令数そのものが少ない
- キャッシュミス率が小さく、キャッシュミス率（キャッシュミス数 */* 全 *commit* 数）の違いとしてはそれほど大きくない。

そこで、各ベンチマークにおける、分岐予測ミス率を図 *10*に、外部キャッシュミス率を図 *11*に示す。

分岐予測ミス率は、*mst* の *3.2%* が最も大きく、*health,espressom,gcc,li* は *2%* 前後である。*art* の分岐予測ミス率は *0.1%* 未満であった。

外部キャッシュミス率は、*art* の *5.5%* が最も大きく、次いで *health* の *1.7%* であった。*espresso,gcc,li* のキャッシュミス率は *0.1%* 未満と非常に小さかった。

キャッシュミス率の高い *art* は分岐予測ミス率が低く、図 *7,* 図 *8*において、*EDS* 方式と *TDS* 方式の *fetch* 命令数に差が見られない。逆に、分岐予測ミス率が高い *mst,espresso,gcc,li* では、*fetch* 命令数、キャッシュミス数に *EDS* 方式と *TDS* 方式の差があるものの、キャッ

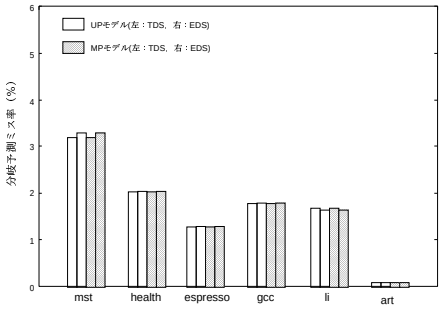

図 10 各ベンチマークの分岐予測ミス率

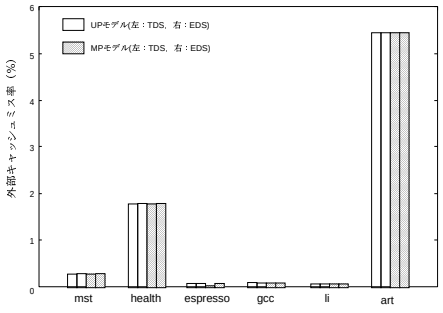

図 11 各ベンチマークの外部キャッシュミス率

シュミス率が低く、キャッシュミス数の違いが実行サイクル数に与える影響は小さいと考えられる。

health に関しては、分岐予測ミス率、キャッシュミス率は、共に大きいが、*EDS* 方式と *TDS* 方式のキャッシュミス数には違いが見られない。*health* では分岐予測外れパスでキャッシュミスが発生しないため、実行サイクル数に与える影響はそれほどないと考えられる。

SPLASH2 において *EDS* 方式と *TDS* 方式の違いが見られない原因も、表 *4* に示す様に分岐予測ミス率、外部キャッシュミス率が小さいためであると考えられる。

表 4 SPLASH2 の特性

	fft	lu	radix	cholesky
分岐予測ミス率	0.28%	0.65%	0.001%	1.6%
キャッシュミス率	0.02%	0.001%	0.001%	0.003%

この様に、分岐予測ミス率、キャッシュミス数が低く、分岐予測外れパスの影響が *CPU* 内で吸収される場合には、マルチプロセッサにおいても、単体 *CPU* と同様に実行サイクル数の差は見られない。

5. ま と め

本論文では、分岐予測外れパスがシミュレーションされるか否かという実行ドリブン型シミュレータ *(EDS)* とトレースドリブン型シミュレータ *(TDS)* の違いに注目し、分岐予測外れパスが計算機システムに及ぼす影響について評価を行った。分岐予測外れパスがシミュレーションされる *EDS* 方式とシミュレーションされない *TDS* 方式のシミュレーション結果の比較を行なったところ、分岐予測外れパスで外部キャッシュミスが頻発するサンプルプログラムにおいて、単一プロセッサシステムで *79%*, マルチプロセッサシステムで *93%* の実行サイクル数の違いが見られ、分岐予測外れパスが性能に大きく影響を与えることを示した。また、*SPEC INT* などのベンチマークを用いた評価の結果、分岐予測外れパスが実行サイクル数に与える影響はそれほど大きくなかった。これは、今回評価に用いたベンチマークが、分岐予測ミス率、外部キャッシュミス率が小さいためであると考えられる。この様に、分岐予測外れパスが実行サイクル数に影響を与えない場合もあるが、分岐予測ミス率、キャッシュミス率が大きい場合には、分岐予測外れパスが実行サイクル数に影響を及ぼす可能性がある。

今回は、分岐予測外れパスの影響として、キャッシュ状態が変化する点に注目したが、ハードウェア資源利用状況の変化も考えられる。資源利用の状況の違いについては、*multi threading* や *SMT(Simultaneous Multi Threading)*[8] といった、複数スレッドでハードウェア資源を共有するアーキテクチャにおいて、より問題となると考えられる。今後は、本論文で開発した *SEEDS* シミュレータを用いて *SMT* アーキテクチャの評価を行なうと同時に、*SMT* アーキテクチャにおける分岐予測外れパスの影響についても調査していく予定である。

参 考 文 献

1) *SMTSIM. http://www-cse.ucsd.edu/users/tullsen/smtsim.html.*
2) *SimpleScalar. http://www.simplescalar.org/.*
3) *L.K.John R.Bhargava and F.Matus. Accurately modeling speeculative instruction fetching in trace-driven simulation.* IPCCC'99, *pp. 65–71, 1998.*
4) *J.D.Wellman M.Moudgill and J.Moreno. An approach for quantifying the impact of not simulating mispredicted paths.* Workshop on Performance Analysis and its Impact on Design, *1998.*
5) *J.H.Reppy A. Rogers, M.C.Carlisle and L.J.Hendren. Supporting dynamic data structures on distributed-memory machines.* ACM Trans. on Programing Languages and Systems, *pp. 233–263, 1995.*
6) *A. Moshovos A.Roth and G.S.Sohi. Dependence based prefetching for linked data structures.* Proc. on ASPLOS-8, *1998.*
7) *Stanford Parallel Application for Shared Memory. http://www-flash.stanford.edu/apps/SPLASH.*
8) *Susan J.Eggers Dean M.Tullsen and Henry M.Levy. Simultaneous multithreading: Maximizing on-chip parallelism.* ISCA'95, *1995.*