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

2005－ARC－165 (1)
2005／11／30

キャッシュ・ミス頻発ロード命令の特徴解析

三輪英樹[†] 堂後靖博[††]
井上弘士[†] 村上和彰[†]

近年，マイクロプロセッサの性能は半導体製造技術の進歩に伴い飛躍的に向上した．一方，主記憶として利用される DRAM は高速化が難しく，その動作周波数はマイクロプロセッサより 2 桁小さい．このため，主記憶がマイクロプロセッサの性能を抑制するという問題 (メモリ・ウォール問題) の解決がコンピュータ・システム性能向上の大きな鍵となっている．筆者らの研究グループではキャッシュ・ミスを頻発させるロード命令に着目し，キャッシュ・ミス・ペナルティを低減する技術を開発中である．キャッシュ・ミス頻発ロード命令は全キャッシュ・ミスの 80 % 以上を発生させ性能へ大きな影響を与える．本稿では，このロード命令によるキャッシュ・ミスの状況を明らかにするために，複数のベンチマーク・プログラムおよび入力データを対象とした調査を行なった．その結果，キャッシュ・ミスは多重ループ内におけるポインタ参照，多次元配列アクセス，もしくは構造体配列アクセス時に発生していることが判明した．また，キャッシュ・ミスが頻発するデータをロードする命令およびストアする命令のいずれも，入力にあまり依存しない場合が多いことが明らかになった．

Characterization Analysis for Delinquent Loads

HIDEKI MIWA,[†] YASUHIRO DOUGO,[††] KOJI INOUE[†]
and KAZUAKI MURAKAMI[†]

In recent years, the performance of microprocessors has been improved extremely. On the other hand, DRAMs, commonly used as the main memory, is about 100 times as slow as microprocessors. In this situation, DRAMs suppress the performance of microprocessors. This problem is commonly called Memory Wall Problem. For the performance improvement of computer systems, it is very important to solve this problem. Currently, the authors are developing cache miss penalty reduction techniques focused on the delinquent loads which cause the cache misses frequently. Such load instructions are responsible for 80% of all the cache misses, and deteriorate the performance. In this paper, to reveal the cause of cache misses, the authors investigate the memory access patterns for several benchmark programs.

1. はじめに

近年，マイクロプロセッサの性能は，主に半導体製造技術の進歩を要因として飛躍的な向上を遂げた．一方，主記憶として利用されている DRAM には，高集積化しやすく単位容量あたりの価格を下げられるという利点があるが，動作周波数を上げにくいという欠点がある．このため，DRAM の動作周波数は 10 数年前はマイクロプロセッサと同等であったが，現在に至るまでに 2 桁も引き離されている．演算が高速化しても，演算で利用されるデータの読み出しが遅ければ処理時間の大幅な短縮は期待できない．すなわち，マイクロプロセッサの性能が DRAM によって抑制されるという状況に陥っている．この問題は **メモリ・ウォール問題** として知られ，問題解決のための様々な研究開発が行なわれてきた．

メモリ・ウォール問題を解決するための 1 手法として，頻繁に参照されるデータをマイクロプロセッサ内部の高速なメモリに記憶する方法 (オンチップ・キャッシュの搭載) が考えられる．この手法はほとんどのマイクロプロセッサに実装されている．しかしながら，オンチップ・キャッシュは容量の制限があるため全てのデータを保持できず，参照されるデータが存在しない状況 (キャッシュ・ミス) が発生する．キャッシュ・ミスが最下層のオンチップ・キャッシュ (ラストレベル・キャッシュ) で発生した場合，オフチップ・メモリ (主記憶) へのアクセスが必要となる．この処理にかかる時間 (ミス・ペナルティ) は非常に長いため，ラストレベル・キャッシュでのキャッシュ・ミス回数の削減もしくはミス・ペナルティの低減がメモリ・システ

† 九州大学大学院システム情報科学府 Dept. of Informatics, Kyushu University
†† 福岡大学大学院工学研究科 Dept. of Electronics Engineering, Fukuoka University Fukuoka University

ム高性能化の鍵となる．

近年の研究により，キャッシュ・ミスの大部分はいくつかのロード命令により引き起されていることが明らかになった．このようなロード命令は，キャッシュ・ミス頻発ロード命令 (**Delinquent Load 命令，以下 DL 命令**) と呼ばれる[3]．DL 命令に着目したメモリ・ウォール問題解決手法としては，ロード対象データのアドレスを先見的に計算しキャッシュ・メモリへプリフェッチする方法が挙げられる[1),3),5),7)]．これらの手法では，DL 命令のロード対象アドレスを先見的に計算して求め，データをプリフェッチすることでミス・ペナルティを低減する．しかしながら，先見的にアドレスを計算するため，求められたアドレスが正しくない場合にはキャッシュ・ミス・ペナルティを低減できない．加えて，プリフェッチにより必要なデータが置い出されることで，逆にキャッシュ・ミスの増加を招く可能性もある．

一方，我々は DL 命令のロード対象データをプリフェッチングもしくはバッファリングすることで，キャッシュ・ミス・ペナルティを低減する手法を開発中である．これまでの研究により DL 命令のロード対象データの多くがプログラム中で計算されたものであることが判明した．すなわち，DL 命令のロード対象データをストアしている命令が存在するということである．我々はこのストア命令を，DL 命令対応ストア命令 (**Corresponding Store 命令，以下 CS 命令**) と呼んでいる．DL 命令および CS 命令の概念を図 1 に示す．他の研究においてもこのようなストア命令が存在するという報告がある[1)]．これまでの研究で，(1) DL 命令にはほぼ CS 命令が存在すること，(2) CS 命令は少数の静的なストア命令であること，の 2 点が明らかになった[8)]．このような DL 命令の特徴を利用することで，ラストレベル・キャッシュでのミス・ペナルティの低減が可能ではないかと考える．しかしながら，これらの命令に関する調査は不十分であり，キャッシュ・ミス・ペナルティを低減可能かどうかについてはより詳しい解析が必要である．

本稿では，DL 命令の特徴を利用したラストレベル・キャッシュ・ミス・ペナルティ低減手法の提案を目的として，DL 命令および CS 命令の特徴をより詳しく解析する．なお，諸般の事情により，本稿では CS 命令および DL 命令を高性能化目的でどのように利用するかという具体的な方法論については言及しない．

本稿は以下のような構成である．まず，第 2 章にて DL 命令および CS 命令を定義し，高性能化に利用する上で調査すべき特徴について整理する．続いて，第 3 章にて DL 命令および CS 命令の特徴を定量的な評価実験から明らかにする．最後に，第 4 章にて本稿をまとめる．

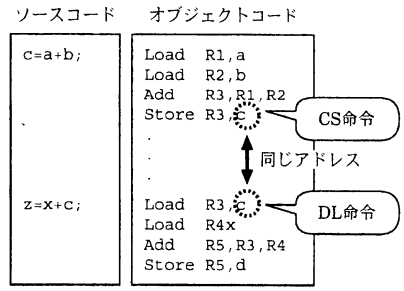

図 1 DL 命令および CS 命令の概念図

2. DL 命令および CS 命令の特徴

本章では，DL 命令および CS 命令を定義し，これまでに判明している特徴についてまとめる．さらに，高性能化に利用するために調査すべき特徴について整理する．

2.1 DL 命令の定義

一般に，DL 命令とはキャッシュ・ミスを頻発させるロード命令であるとされているが，その共通の定義は存在しない．これまで，我々は DL 命令をキャッシュ・ミスを多く発生させる上位 16 個のロード命令と定義していたが，根拠がなく不適切であった．本稿では，DL 命令を **ラストレベル・キャッシュ・ミスを発生させた静的なロード命令のうち，キャッシュ・ミスの 80% を占めるロード命令** と定義する．つまり，最もミス回数が多いロード命令から順にミス回数を加えていき，占有率が 80% 以上になったロード命令までを DL 命令と定義する．以下，この定義を採用する理由について説明する．

ラストレベル・キャッシュ・ミスを対象としている点については，キャッシュ・ミス時にはオフチップ・アクセスが発生し，性能への悪影響が大きいためである．ロード命令によるキャッシュ・ミスに限定している点については，ストア時のミスは既存技術により性能に与える影響がほとんどないことが理由である．

DL 命令が占めるキャッシュ・ミス率を 80% とした点については，シミュレーション実験の結果を元に決定した．実験環境は第 3 章での評価実験のものと同様である．実験では，複数のベンチマークプログラムに 2 種類の入力データ (train および test) を与え，ラストレベル・キャッシュ・ミスの 90% もしくは 80% を占める静的なロード命令数および割合を求めた．結果

表 1　90% もしくは 80% のキャッシュ・ミスを起こす静的なロード命令数および割合

ベンチマーク・プログラム名	入力データ	静的なロード命令数 (割合) 90%		80%		全ロード命令数
168.wupwise	train	7	(3%)	5	(2%)	208
	test	7	(4%)	5	(3%)	194
171.swim	train	21	(8%)	18	(7%)	263
	test	19	(8%)	17	(7%)	230
172.mgrid	train	12	(4%)	8	(3%)	271
	test	12	(4%)	9	(3%)	271
173.applu	train	12	(5%)	9	(4%)	236
	test	24	(15%)	14	(9%)	164
177.mesa	train	1	(0%)	1	(0%)	309
	test	2	(1%)	2	(1%)	266
179.art	train	2	(3%)	1	(2%)	63
	test	1	(2%)	1	(2%)	56
183.equake	train	23	(8%)	17	(6%)	274
	test	26	(10%)	19	(7%)	263
188.ammp	train	15	(3%)	7	(1%)	593
	test	5	(1%)	2	(0%)	564
301.apsi	train	7	(4%)	7	(4%)	186
	test	14	(7%)	7	(4%)	189
164.gzip	train	2	(1%)	1	(0%)	215
	test	1	(1%)	1	(1%)	161
175.vpr_place	train	3	(2%)	2	(2%)	121
	test	17	(15%)	5	(5%)	110
175.vpr_route	train	12	(2%)	6	(1%)	544
	test	17	(7%)	9	(4%)	231
176.gcc	train	363	(9%)	161	(4%)	3926
	test	380	(19%)	168	(9%)	1953
181.mcf	train	10	(3%)	6	(2%)	322
	test	13	(15%)	2	(2%)	89
197.parser	train	21	(2%)	6	(1%)	1049
	test	31	(5%)	12	(2%)	584
256.bzip2	train	28	(9%)	20	(6%)	317
	test	11	(5%)	7	(3%)	224

を表 1 に示す．表 1 は左のカラムから順に，ベンチマーク・プログラム名，入力データ，キャッシュ・ミスの 90% もしくは 80% を占める静的なロード命令数および割合，ミスを発生させた静的な全ロード命令数を表わしている．キャッシュ・ミスの 90% もしくは 80% を占める静的なロード命令数を比較すると，90% ではロード命令数が一部で非常に多くなっていることがわかる．我々が開発中の高性能化技術はハードウエアによる実装を考えており，対象とする静的なロード命令数が少ないほうが追加で必要となるハードウエアを少なく抑えられる．この観点から，80% のキャッシュ・ミスの発生に関わっている静的なロード命令を DL 命令と定義することとした．

2.2 DL 命令および CS 命令の特徴

これまでの研究により，DL 命令によるロード時にキャッシュ・ミスが発生するデータの多くは，それ以前にストア命令に書き込まれたものであるということが明らかになっている．我々は，このようなストア命令を CS 命令と呼んでおり，また Annavaram らの研究グループは我々より若干早く Problem Store として発表している[1]．DL 命令によるキャッシュ・ミスに対処するためには，DL 命令によりロードされるデータの値もしくはアドレスを知る必要があるが，CS 命令はこの情報を持っている．したがって，CS 命令をうまく利用することにより DL 命令によるキャッシュ・ミスを予測もしくは予防することが可能になるのではないかと考えている．

我々は，この観点から CS 命令および DL 命令の特徴の調査を行ってきた[8]．これまでに判明しているのは以下の 2 点である．

(1) キャッシュ・ミス率が高いベンチマーク・プログラムにおいて DL 命令がキャッシュ・ヒットすると仮定した場合，非常に大きな性能向上が得られること．

(2) CS 命令は，DL 命令に対してほぼ 100% の確率で存在すること．

(3) DL 命令数および CS 命令数はいずれも少数である．

まず上記 (1) は，ラストレベル・キャッシュにおけるミス・ペナルティが実行時間の大半を占めていることを示している．したがって，メモリ・システムの高性能化のためには，キャッシュ・ミスの大半を引き起こしている DL 命令をキャッシュ・ヒットさせる必要があるということである．次に，上記 (2) は，DL 命令と CS 命令が非常に密接に関係していることを暗示している．しかしながら，この関係についてはまだ明らかになっていないため，早急に解明する必要がある．最後に，上記 (3) は，これらの命令を高性能化に利用する上で非常に有利な特徴である．ただし，あくまでも静的な命令数であるため，多数のインスタンス (動的な命令) が存在することに注意しながら高性能化を考えなければならない．

2.3 調査すべき DL 命令および CS 命令の特徴

第 2.2 節で，DL 命令および CS 命令の特徴についてこれまでに判明しているものを示した．しかしながら，これまでに我々が実施した調査では，以下のような問題がある．

- DL 命令の定義が不適切であった．これに関しては既に第 2.1 節にて再定義した．
- ベンチマーク・プログラムへの入力が変化した場合を考慮していない．
- DL 命令および CS 命令について，プログラム中での処理との関連が未調査である．
- DL 命令および CS 命令の特徴解析はシミュレーション実験に基づいているが，シミュレーションで実行した命令数が不十分であった．

以上より，DL 命令および CS 命令を高性能化に利用する際には，これまでの調査に加えて以下の項目に関する調査が必要であると考える．

- **DL 命令が全てキャッシュ・ヒットすると仮定した場合の性能向上可能性，およびその入力依存性**: DL 命令の定義を変更したため，再度 DL 命令のキャッシュ・ミスを予防できた場合にどの程度の性能向上が得られるかを調査する．
- **CS 命令の存在状況，およびその入力依存性**: CS 命令が存在することは明らかになっているが，入力が変化した場合も存在するのかを調査する．
- **DL 命令および CS 命令の対応状況，およびその入力依存性**: ある CS 命令が書き込んだデータをある DL 命令が読み込むという関係が，入力によってどのように変化するか調査する．
- **DL 命令および CS 命令と実際の処理との関係，および入力依存性**: DL 命令および CS 命令は実際のプログラム上でどのようなデータ構造に対してアクセスしているのかを調べ，既存技術での対応ができるかどうかについて検討する必要がある．

以上の調査項目について，第 3 章で定量的な評価実験を通じて考察する．

3. 検討事項に対する考察

本章では，第 2 章にて挙げた検討事項について，定量的な評価実験の結果をもとに考察する．

3.1 シミュレーション環境環境

評価実験は，マイクロプロセッサシミュレータである SimpleScalar Toolset[2] Version 3.0d sim-outorder を利用した．本評価実験で利用したシミュレータのプロセッサ構成に関する主なパラメータを表 2 に示す．

評価対象ベンチマーク・プログラムは，SPEC CPU 2000 ベンチマークセット[4] より表 3 に示す 16 種類を対象とした．これらのベンチマークプログラムは，SimpleScalar に対応した gcc-2.7.2.3 により，'-02' オプション付きでコンパイルした[6]．

各ベンチマーク・プログラムへの入力データとしては，SPEC CPU 2000 ベンチマークセットに含まれている Train および Test を利用し，全命令を実行した．

3.2 DL 命令が全てキャッシュ・ヒットすると仮定した場合の性能向上可能性

まず，全ての DL 命令がキャッシュ・ヒットすると仮定した場合，どの程度の性能向上が得られるか調査した．結果を表 4 に示す．表 4 は左のカラムから順に，ベンチマーク・プログラム名，入力データ，キャッシュ・ミス率，全 DL 命令キャッシュ・ヒット仮定時の実行時間削減率を表している．この結果よりベンチマーク・プログラムは以下の 3 種類に分類できる．

表 2 シミュレータ設定

命令発行方式	アウト・オブ・オーダ
命令セット	PISA
分岐予測器	
Type	2 レベル (gshare, 2K エントリ)
BTB サイズ	512 エントリ，4 ウェイ
RAS	32
命令発行幅	8 命令/cc
命令デコード幅	8 命令/cc
IFQ サイズ	8 エントリ
RUU サイズ	64 エントリ
LSQ サイズ	32 エントリ
キャッシュ・メモリ	
L1 データキャッシュ	32KB (64B/ エントリ，2 ウェイ，256 エントリ)
L1 命令キャッシュ	32KB (64B/ エントリ，1 ウェイ，512 エントリ)
L2 共有キャッシュ	2MB (64B/ エントリ，4 ウェイ，8192 エントリ)
レイテンシ	
L1 キャッシュ	1 cc
L2 キャッシュ	16 cc
主記憶	250 cc
メモリバンド幅	8B
メモリポート数	2
ITLB, DTLB	
エントリ	1M エントリ (4KB/ エントリ，256 エントリ/ ウェイ，4 ウェイ)
ミスペナルティ	30 cc
整数演算器 (ユニット数，実行，発行レイテンシ)	
ALU	4, 1 cc, 1 cc
Mult.	1, 3 cc, 1 cc
Div.	1, 20 cc, 19 cc
浮動小数点演算器 (ユニット数，実行，発行レイテンシ)	
ALU	4, 2 cc, 1 cc
Mult.	1, 4 cc, 1 cc
Div.	1, 12 cc, 12 cc
SQRT	1, 24 cc, 24 cc

(cc: clock cycle (s))

表 3 評価実験で利用するベンチマーク・プログラム

ベンチマーク名	プログラムのカテゴリ
浮動小数点演算系	
168.wupwise	量子化学
171.swim	波面モデリング
172.mgrid	3 次元電界
173.applu	楕円幾何学
177.mesa	3 次元図形処理ライブラリ
179.art	画像認識，ニューラルネットワーク
183.equake	地震波伝播シミュレーション
188.ammp	計算化学
301.apsi	気象学
整数演算系	
164.gzip	圧縮
175.vpr_place	回路配置
175.vpr_route	回路配線
176.gcc	C 言語コンパイラ
181.mcf	組合せ最適化
197.parser	文字列処理
256.bzip2	圧縮

(1) *168.wupwise*, *171.swim*, *172.mgrid*, *183.equake*, *188.ammp*, *256.bzip2*: 入力に依らず実行時間削減率が高い．これは，いずれの入力であってもロード命令によるキャッシュ・ミス率が高く，その 80% がヒットに変化することでキャッシュ・ミス・ペナルティが大幅に低減できるためである．

(2) *173.applu*, *175.vpr_route*, *181.mcf*,

表 4　キャッシュ・ミス率および DL 命令キャッシュ・ヒット仮定時の実行時間削減率

ベンチマーク・プログラム名	入力データ	キャッシュ・ミス率 ロード	ストア	合計	実行時間削減率
168.wupwise	train	18%	4%	22%	34%
	test	15%	5%	20%	30%
171.swim	train	32%	11%	42%	68%
	test	39%	17%	56%	66%
172.mgrid	train	31%	10%	42%	51%
	test	36%	10%	45%	54%
173.applu	train	26%	30%	56%	30%
	test	1%	1%	2%	2%
177.mesa	train	2%	31%	33%	3%
	test	0%	4%	5%	2%
179.art	train	0%	0%	0%	0%
	test	0%	0%	0%	0%
183.equake	train	58%	2%	60%	43%
	test	33%	3%	36%	29%
188.ammp	train	31%	4%	36%	56%
	test	53%	2%	55%	62%
301.apsi	train	0%	1%	1%	0%
	test	0%	2%	2%	0%
164.gzip	train	1%	1%	2%	2%
	test	1%	2%	2%	3%
175.vpr_place	train	0%	0%	0%	0%
	test	0%	0%	0%	0%
175.vpr_route	train	5%	0%	5%	27%
	test	0%	0%	0%	0%
176.gcc	train	0%	0%	1%	3%
	test	0%	0%	0%	0%
181.mcf	train	7%	3%	10%	32%
	test	0%	15%	15%	0%
197.parser	train	2%	2%	4%	9%
	test	0%	0%	1%	1%
256.bzip2	train	6%	4%	10%	18%
	test	7%	6%	13%	19%

197.parser: 入力によって実行時間削減率が大幅に異なる．これは，入力によりロード命令によるキャッシュ・ミス率が大幅に変化したためである．

(3) その他のベンチマーク・プログラム: キャッシュ・ミス率が低く，メモリ・ウォール問題が存在していない．このようなベンチマーク・プログラムでは実行時間のうち演算が占める割合が高いため，演算時間を短縮するような手法により性能向上が可能であると考える．

以降，上記 (1),(2) に分類したベンチマーク・プログラムについて考察する．

3.3 CS 命令の存在状況

次に， DL 命令に対して CS 命令が存在しているかどうか調査した．結果を図 2 に示す．横軸はベンチマーク・プログラム、縦軸は DL 命令に対して CS 命令が存在した割合を示している。また、2 本のバーは左側が Train 入力を利用した場合で右側が Test 入力を利用した場合である。*175.vpr_route*, *181.mcf* で test 入力を利用した場合に CS 命令の存在率が低くなっているが，それ以外のベンチマーク・プログラムでは常に存在することがわかる．したがって，多くの場合に CS 命令を利用して DL 命令のロード対象データの値およびアドレスを知ることが可能であると言える．

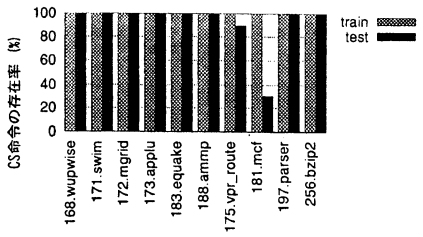

図 2　CS 命令の存在する割合

表 5　DL 命令数および CS-DL ペア数

ベンチマーク・プログラム名	DL 命令数 train	test	共通	CS-DL ペア数 train	test	共通
168.wupwise	5	5	5	19	19	19
171.swim	18	17	17	50	48	48
172.mgrid	8	9	7	35	40	32
173.applu	9	14	6	64	46	35
183.equake	17	19	17	25	28	25
188.ammp	7	2	2	12	3	3
175.vprroute	6	9	1	28	39	3
181.mcf	6	2	1	39	4	4
197.parser	6	12	4	37	48	28
256.bzip2	20	7	7	82	37	37

3.4 DL 命令および CS 命令の対応状況

続いて， DL 命令と CS 命令とはどのように対応しているのかを調査した．具体的には， DL 命令実行時にキャッシュ・ミスを起こしたデータを仲介役として DL 命令と CS 命令とを “ペア” ととらえ，このペア数がいくつあるかを調査した．結果を表 5 に示す．表 5 は左のカラムから順に，ベンチマーク・プログラム名，DL 命令数， CS-DL ペア数を表している．また，DL 命令数および CS-DL ペア数については 3 つのカラムからなり， train 入力利用時， test 入力利用時，入力に依存せず共通して出現する命令もしくはペアの数を表している．

表 5 より，*188.ammp*, *175.vpr_route*, *181.mcf* は入力が test の場合のペア数が少なく，結果的に入力に依存せず出現するペアの数が少なくなっている．この理由は，表 5 より入力に依存せず出現した DL 命令の数が少ないためである．これら以外のベンチマーク・プログラムでは，入力に依存しない CS-DL ペアが数多く出現していることがわかる．このような DL 命令に対しては，コンパイル時に CS-DL ペアの情報を反映したコードを生成することでキャッシュ・ミスを防ぐことができる可能性がある．

3.5 DL 命令および CS 命令とプログラム中での処理との関係

最後に， DL 命令および CS 命令は実際のプログラム・コード中でどのような処理をおこなっているの

表 6 DL 命令のプログラム中での処理

ベンチマーク・プログラム名	入力データ	アクセス対象データ構造別 DL 命令数
168.wupwise	train	構造体配列:4, ポインタ:1
	test	構造体配列:4, ポインタ:1
171.swim	train	多重ループ内構造体配列:18
	test	多重ループ内構造体配列:17
172.mgrid	train	多重ループ内配列:8
	test	多重ループ内配列:9
173.applu	train	多重ループ内配列:9
	test	多重ループ内 { 配列:7, 構造体配列:7 }
183.equake	train	多重ループ内多次元配列:17
	test	多重ループ内多次元配列:19
188.ammp	train	構造体配列 (ポインタ要素):1, ポインタ:4
	test	ポインタ:2
175.vpr_route	train	ループ内構造体配列:4, 多重ループ内多次元配列:1, ポインタ:1
	test	ループ内構造体配列:3, 多重ループ内ポインタ配列:2, 多重ループ内多次元配列:1
181.mcf	train	ループ内ポインタ:6
	test	ループ内ポインタ:1, その他:1
197.parser	train	{ ループ内, 再帰関数内 } ポインタ:6
	test	{ ループ内, 再帰関数内 } ポインタ:10, ループ内配列:2
256.bzip2	train	ループ内配列:19, 構造体配列 (配列要素):1
	test	ループ内配列:7

かについて調査した．これにより，高性能化手法を評価する際に，どのような既存技術との比較が必要であるかが判明する．結果を表 6 に示す．表 6 は左のカラムから順に，ベンチマーク・プログラム名，入力，およびアクセス対象データ構造別で分類した DL 命令数を表している．表 6 には DL 命令がアクセスしているデータ構造のみを記しているが，CS 命令は同様のデータ構造に対して書き込みをおこなうことから割愛した．

どのベンチマーク・プログラムおよび入力にも共通しているのが，ループ内で配列もしくはポインタに対する処理をおこなっていることである．第 3.4 節にて CS-DL ペアは入力に依存しないものが多いことを述べたが，このことからコンパイル時にループを分割する手法 (ループのブロッキング) によりこれらのキャッシュ・ミスを防げる可能性がある． DL 命令および CS 命令を利用した高性能化手法を評価する際には，ブロッキングとの比較が必須となると考える．

4. おわりに

本稿では，メモリ・システムの高性能化手法を開発するにあたり， DL 命令および CS 命令についてより詳しい調査をおこなった．評価実験の結果， DL 命令および CS 命令は入力によって大きく変化する場合は少なく， DL 命令によるキャッシュ・ミスを防止できれば最大 70% 近く実行時間を削減できることが判明した．また， DL 命令および CS 命令は，ループ中の配列もしくはポインタに対してデータを操作する場合が多いということも示された．今後，我々は本稿では示していない DL 命令および CS 命令の特徴を調査した上で高性能化手法を確定し，その性能に与えるインパクトを明らかにする予定である．

謝辞 本研究を進めるにあたり，多くのご指導を頂いた安浦寛人教授をはじめとする研究室諸氏，富士通株式会社の池田正幸氏および丸山拓巳氏，富士通研究所の勝野昭氏および坂本真理子氏に深く感謝致します．本研究の一部は， 21 世紀 COE プログラム「システム情報科学での社会基盤システム形成」および文部省科学研究費補助金 (課題番号: 17680005) による補助のもとで行なわれた．

参考文献

1) Annavaram, M., Patel, J. M. and Davidson, E. S.: Data Prefetching by Dependence Graph Precomputation, *Proc of the 28th Intl. Symposium on Computer Architecture* (2001).
2) Burger, D. and Austin, T. M.: The SimpleScalar Tool Set, Version 2.0, *University of Wisconsin-Madison Computer Sciences Department Technical Report* (1997).
3) Collins, J. D., Wang, H., Tullsen, D. M., Hughes, C., Hoflener, G., Lavery, D. and Shen, J. P.: Speculative Precomputation: Long-range Prefetching of Delinquent Loads, *Proc of the 28th Intl. Symposium on Computer Architecture* (2001).
4) Henning, J. L.: SPEC CPU2000: Measuring CPU Performance in the New Millennium, Vol. 33, No. 7 (2000).
5) Moshovos, A., Pnevmatikatos, D. N. and Baniasadi, A.: Slice-Processors, *Proc of the Intl. Conference on Supercomputiong* (2001).
6) Oskin, M.: PISA GCC 2.7.2.3 Cross Compiler. http://arch.cs.ucdavis.edu/RAD/.
7) Roth, A. and Sohi, G.: Speculative Data-Driven Multithreading, *Proc of the 7th Intl. Symposium on High-Performance Computer Architecture* (2001).
8) 三輪英樹, 堂後靖博, 井上弘士, 村上和彰: キャッシュ・ミス頻発ロード命令を対象としたミス原因解析, 電子情報通信学会技術研究報告, CPSY2005-22 (2005).