九州大学学術情報リポジトリ
Kyushu University Institutional Repository

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

三輪, 英樹
九州大学大学院システム情報科学府

堂後, 靖博
福岡大学大学院工学研究科

井上, 弘士
九州大学大学院システム情報科学府

村上, 和彰
九州大学大学院システム情報科学府

https://hdl.handle.net/2324/6238

出版情報：電子情報通信学会技術研究報告, CPSY2005-22(2005-08). 105 (226), pp.43-48, 2005-08. 電子情報通信学会CPSY研究会
バージョン：
権利関係：

KYUSHU UNIVERSITY

社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

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

三輪　英樹†　　堂後　靖博††　　井上　弘士†　　村上　和彰†

† 九州大学大学院システム情報科学府
†† 福岡大学大学院工学研究科
E-mail: †{h-miwa,inoue,murakami}@i.kyushu-u.ac.jp, ††dougo@v.tl.fukuoka-u.ac.jp

あらまし　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．その一方で，主記憶として利用される DRAM は構造的に高速化しにくく，今やその速度はマイクロプロセッサよりも約 100 倍遅い．このような状況では，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る．現在，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てキャッシュ・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・ミスの約 90 % 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・ミスの原因を明らかにするために，複数のベンチマーク・プログラムに関してメモリ・アクセス・パタンの調査を行なった．その結果，キャッシュ・ミス・頻発ロード命令のロード対象データの殆どは，プログラム実行中に一旦ストアされたデータであることが判明した．

キーワード　キャッシュ・メモリ，メモリ・ウォール問題，キャッシュ・ミス・ペナルティ低減，キャッシュ・ミス頻発ロード命令

Behavior Analysis for Delinquent Loads

Hideki MIWA†, Yasuhiro DOUGO††, Koji INOUE†, and Kazuaki MURAKAMI†

† Dept. of Informatics, Kyushu University
†† Dept. of Electronics Engineering, Fukuoka University
E-mail: †{h-miwa,inoue,murakami}@i.kyushu-u.ac.jp, ††dougo@v.tl.fukuoka-u.ac.jp

Abstract　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 90% 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. The results show that almost all of the data which cause cache misses had been written to memory system by store instructions.

Key words　cache memory, memory wall problem, cache miss penalty reduction, delinquent load instructions

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，動作速度を上げにくいという欠点がある．このため，約 25 年前はほぼ同じであった両者の動作速度は，今や 100 倍以上 DRAM が遅くなっている．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結果として演算にかかる時間は短縮されない．このような状況ではマイクロプロセッサの性能が DRAM の性能によって抑制される．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が考えられる．既存手法としては，頻繁に参照されるデータをマイクロプロセッサ内部にキャッシュしておくためのオンチップ・キャッシュ・メモリが挙がる．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，全てのデータをキャッシュ・メモリに格納することはできず，参照されたデータがキャッシュ・メモリに存在しない状況 (キャッシュ・ミス) は発生する．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 [1].

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]〜[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キャッシュ・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間違いである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た場合，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 命令に着目したメモリ・ウォール問題解決手法としてロード対象データを再計算する方法を提案および評価した [8]．本手法の目的は，DL 命令を実行する代わりにロード対象データを再計算して求めることで，キャッシュ・ミス・ペナルティ低減効果を得ることにある．残念ながらこの手法を実現するためには非現実的な仮定が必要となるため，研究を断念している．しかしながら，この研究の過程において 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．DL 命令と CS 命令との間に何らかの特徴がある場合，その特徴を利用することで DL 命令によるキャッシュ・ミス・ペナルティを低減できるのではないかと考えている．そこで，本稿では，キャッシュ・ミス・ペナルティ低減手法の提案のための予備検討として，DL 命令および CS 命令に着目することで性能向上効果が得られるかどうかを検討する．なお，本稿では，CS 命令を性能向上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考慮しない．

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，これらの命令を性能向上に活用することの妥当性を示すための検討項目について整理する．続いて，第 3. 章にて各検討項目に関して定量的な評価実験から実態を明らかにする．最後に，第 4. 章にて本稿をまとめる．

2. DL 命令および CS 命令の定義および検討項目

本章では，DL 命令および CS 命令の定義について述べ，これらを性能向上に活用することの妥当性を示すために必要となる検討項目を整理する．

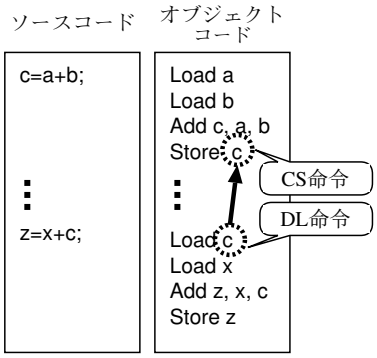

図 1　DL 命令および CS 命令の概念図

なお，問題の簡単化のため以下のようなマイクロプロセッサ構成でのプログラム実行を仮定する．

- 32bit の RISC 型マイクロプロセッサである．
- スレッドは 1 つ．
- キャッシュ・メモリは 2 階層搭載されている．L2 キャッシュにてデータ・キャッシュ・ミスが発生した場合，オフチップ・メモリである主記憶へのアクセスが発生する．

2.1 DL 命令および CS 命令の定義

一般的に，DL 命令とはキャッシュ・ミスを頻発させるロード命令を指す．しかしながら，その明確な定義はない．以下で DL 命令および CS 命令を定義する．

プログラムコードにおけるアドレスがそれぞれ pc_i ，pc_j であるようなロード命令およびストア命令を，式 (1) および式 (2) のように表記する．

$$Load(pc_i) \qquad (i = 1, 2, \cdots, i_{load}) \tag{1}$$

$$Store(pc_j) \qquad (j = 1, 2, \cdots, j_{store}) \tag{2}$$

- pc_i, pc_j: ロード命令およびストア命令のアドレス．
- i_{load}, j_{store}: 全ロードおよび全ストア命令数．

さらに，第 n, m 回目に実行された命令 (第 n, m 番目のインスタンス) を，式 (3) および式 (4) のように表記する．

$$Load(pc_i, n) \qquad (n = 1, 2, \cdots, n_i) \tag{3}$$

$$Store(pc_j, m) \qquad (m = 1, 2, \cdots, m_j) \tag{4}$$

- n, m: インスタンス番号．
- n_i, m_j: pc_i, pc_j のロード命令およびストア命令に対する総インスタンス数．

一般的に，式 (1) および式 (2) の命令は静的，式 (3) および式 (2) の命令は動的な命令と呼ばれる．

動的なロード命令もしくはストア命令 d_inst のアクセス対象データアドレス $addr$ を，式 (5) のように表記する．さらに，動的なロード命令の L2 データ・キャッシュ・ミス状況 $stat$ を，式 (6) のように表記する．

$$addr = DataAddr(d_inst) \tag{5}$$

$$stat = L2DataMissStatus(Load(pc_i, n)) \tag{6}$$

(ただし，キャッシュ・ヒット時： stat=1，ミス時: stat=0)

静的なロード命令のキャッシュ･ミス回数 $L2DataMissCount\left(Load(pc_i)\right)$ は，動的なロード命令のキャッシュ・ミス状況から求められる．

$$L2DataMissCount\left(Load(pc_i)\right) = \sum_{n=1}^{n_i} L2DataMissStatus\left(Load(pc_i, n)\right) \tag{7}$$

静的な各ロード命令について L2 キャッシュ・ミス回数 $L2DataMissCount\left(Load(pc_i)\right)$ を求め，多いものから順に並べて 1 から始まるランクを付ける．ランクは式 (8) で表記する．

$$L2DataMissRank\left(Load(pc_i)\right) \tag{8}$$

以上から DL 命令および CS 命令が定義できる．なお，本稿では DL 命令を L2 データ・キャッシュ・ミスを発生させる上位 16 個の静的なロード命令と定義する(注1)．また，CS 命令と DL 命令とでデータ型は常に一致すると仮定する．このとき，式 (9) を満たす $dlpc_x$ をもつ静的なロード命令が DL 命令であり，式 (10) を満たす $cspc_y$ を持つ静的なストア命令が CS 命令である．

$$L2DataMissRank\left(Load(dlpc_x)\right) \leqq 16 \tag{9}$$

$$DataAddr\left(Load(dlpc_x, n)\right) = DataAddr\left(Store(cspc_y, m)\right) \tag{10}$$

(ただし，n, m は $n \leqq n_{pc_i}, m \leqq m_{pc_j}$ を満たす任意の整数)

2.2 DL 命令および CS 命令に着目したキャッシュ・ミス・ペナルティ低減手法の妥当性を示すための検討項目

我々は，DL 命令によるキャッシュ・ミス・ペナルティを低減することによりメモリ・システムの性能を向上させようと考えている．そこで着目しているのが CS 命令である．DL 命令と CS 命令との間に何らかの特徴がある場合，その特徴を利用することで DL 命令によるキャッシュ・ミス・ペナルティを低減できるのではないかと考えている．

しかしながら，DL 命令および CS 命令の具体的な活用方法を検討する前に，これらの命令に着目することの妥当性を検討する必要がある．本節では，この妥当性を示すための検討項目を挙げる．検討すべき項目は，以下の 3 点である．

まず，DL 命令によるキャッシュ・ミスがどの程度の性能低下を招いているのかという点を検討する必要がある．換言すれば，DL 命令によるキャッシュ・ミスを解消したとして，どれだけ性能が向上するのかということである．この影響が小さい場合，いかなる手法を考案したとしても DL 命令に着目することは意味がない．

次に，CS 命令が存在するのかという点について検討する必要がある．CS 命令が存在する場合，DL 命令が実行されるよりも前の時点でロード対象データの値もしくはアドレスを知ることができる．しかしながら，CS 命令が存在しない場合は我々のアイデアを適用してメモリ・システムの性能を向上させることができない．できるだけ多くの DL 命令に対して CS 命令が発見されることが望ましい．

最後に，DL 命令と CS 命令とがどのように対応しているのかを調査する必要がある．基本的には，少数もしくは多数の CS 命令がストアした値を DL 命令がロードする状況が考えられる．CS 命令をどのように性能向上のために利用するかという具体的な方法論を検討するに際しては，少数の CS 命令を対象にするほうが手間が少ない可能性がある．したがって，どのように DL 命令と CS 命令との対応関係の調査が必要である．

以上をまとめると検討事項は以下の 3 つになる．

- DL 命令のキャッシュ・ミスを解消することで，プログラム全体の実行時間は削減できるのか．
- CS 命令は存在するのか．
- CS 命令と DL 命令の対応関係はどのようになっているのか．

この 3 つの事項について，第 3. 章にて定量的な評価実験を通して考察を行なう．

3. 検討事項に対する考察

本章では，第 2. 章にて挙げた 3 つの検討事項について，定量的な評価実験の結果をもとに考察する．

3.1 評価環境

評価実験は，プログラム実行型マイクロプロセッサシミュレータである SimpleScalar Toolset [9] Version 3.0d から，アウト・オブ・オーダ実行可能な `sim-outorder` を利用した．SimpleScalar は，学術研究分野で幅広く利用されているマイクロプロセッサシミュレータであり，命令レベルのシミュレーションが行なえる．命令セットとしては，32bit 版 MIPS [10] 互換の PISA もしくは Alpha AXP [11] のいずれかから選択可能である．本実験では，第 2. 章の冒頭で述べた前提に基づき，PISA を採用している．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 1 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 [12] より表 2 に示す 10 種類を対象とした．これらのベンチマークプログラムは，SimpleScakar に対応した gcc-2.7.2.3 により，‘`-O2`’ オプション付きでコンパイルした [13]．さらに，コンパイル後のバイナリ・コードが正常に動作することを確かめるため，SimpleScalar に含まれる機能シミュレータ `sim-safe` にて各プログラムを実行した．このようにして得た実行結果と，SPEC CPU 2000 に添付されている実行結果とを比較して同一であることを確かめた．

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入力セットのうち，Reference 入力セットを利用した．ただし，この入力セットを利用すると実行命令数が非常に多くなるため，シミュレータのオプションで実行命令数を制限した．具体的には，先頭の 20 億命令については機能シミュレーションのみを行ない，後続

(注1)：ただし，上位 16 個に限定している根拠は特になく，再度検討し直す必要がある．

表 1　シミュレータ設定

命令発行方式	アウト・オブ・オーダ
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 2　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	処理内容
浮動小数点演算系ベンチマーク	
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
整数演算系ベンチマーク	
164.gzip	圧縮
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
255.vortex	オブジェクト指向データベース
256.bzip2	圧縮

の 2 億命令についてのみ詳細なシミュレーションを行なった．

3.2　DL 命令が性能に与える影響

まず，DL 命令が性能に与える影響を調査した．具体的には，各ベンチマーク・プログラムをオリジナルのシミュレータで実行した場合，および DL 命令による L2 ミス時にキャッシュ・ヒットを仮定した場合の実行時間を求めた．全 DL 命令がキャッシュ・ヒットすると仮定した場合の実行時間削減率を図 2 に示す．

図 2 により，以下のことが言える．

- *179.art*, *188.ammp*, *181.mcf*: 全 DL 命令がキャッシュ・ヒットした場合，実行時間が 80-90% 程度削減できるという結果が得られた．これらのベンチマーク・プログラムでは DL 命令が実行時間に与える影響が大きいことから，DL 命令に着目した対策を施すことで大きなキャッシュ・ミス・ペナルティ低減効果が得られると期待できる．
- *183.equake*, *164.gzip*, *197.parser*, *255.bzip2*: 先の 3 つのベンチマーク・プログラム程ではないが，実行時間が 10-30% 削減できるという結果が得られた．大きな効果は期待できないが，一定の効果を得られる可能性がある．
- *177.mesa*, *176.gcc*, *255.vortex*: 実行時間削減率は 10% 未満である．これらのベンチマーク・プログラムには，どのような手法を適用してもあまり高い効果は得られないと言える．

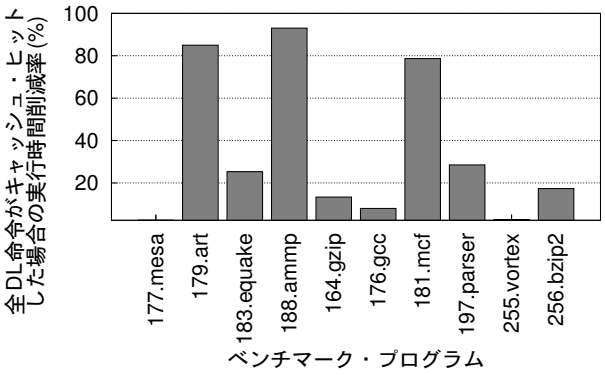

図 2　全 DL 命令キャッシュ・ヒット時の実行時間削減率

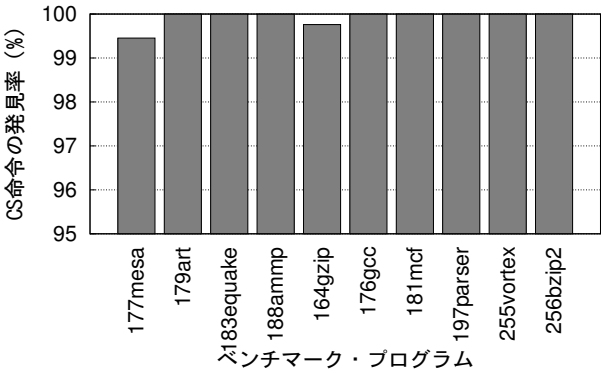

図 3　CS 命令の発見率

3.3　CS 命令の存在状況

次に，DL 命令が判明した時に CS 命令が存在するかどうかを調査した．すなわち，あらかじめ DL 命令の PC が与えられた場合，DL 命令による L2 キャッシュ・ミスを発生時にロード対象データをストアした命令があるかどうかを実験により調べた．DL 命令のキャッシュ・ミス回数に対する CS 命令の発見数の割合を図 3 に示す．

図 3 により，全てのベンチマーク・プログラムにおいて DL 命令のロード対象データは CS 命令によってストアされたことがわかる．この結果から，DL 命令のキャッシュ・ミス・ペナルティを低減する目的で，CS 命令に着目することに意味があると言える．

3.4　CS 命令と DL 命令との対応状況

第 3.3 節の実験では，DL 命令と CS 命令との間には関係があることは判明したものの，それらがどのように対応しているのか，および CS 命令は何命令存在するのかといった情報はわからない．我々は，DL 命令と CS 命令とがどのように対応しているのかについて調査した．本節では，第 3.2 節において高い実行時間削減率が示された 3 つのベンチマーク・プログラム (*179.art*, *188.ammp*, *181.mcf*) について，調査結果を示す．

図 4, 5, 6, に，各ベンチマーク・プログラムの DL 命令およ

び CS 命令の対応状況を示す．図中の左列に DL 命令の PC を，右列に CS 命令の PC を記している．また，DL 命令の PC の左側に各 DL 命令の L2 キャッシュ・ミス回数ランキングの順位を “DL” の文字に続けて表記している．左列から右列へ向って伸びる矢印により，DL 命令が参照したデータがどの CS 命令によりストアされたものであるかを示している．

例えば，図 5 の最初のエントリは以下のような意味である．

- *188.ammp* において第 16 番目に L2 キャッシュ・ミスを多く発生させた DL 命令は，アドレス (PC) が 401028 であり，キャッシュ・ミスを 2 回発生させている．
- 2 回のキャッシュ・ミス時に参照されたデータは，アドレス (PC) が 400f48 であるようなストア命令によりストアされたものである．

第 3.2 節において高い実行時間削減率が示された 3 つのベンチマーク・プログラム (*179.art*, *188.ammp*, *181.mcf*) について図 4, 5, 6 から，以下のことが言える．

- *179.art*: 多くのキャッシュ・ミスは，アドレス (PC) が 400bb8, 401d90 のストア命令によってストアされたデータの参照時に発生したことがわかる．これらのストア命令に着目した対策を施すことでメモリ・システムを高性能化できる可能性がある．
- *188.ammp*: ほとんどのキャッシュ・ミスは，アドレス (PC) が 40ad98 のストア命令によってストアされたデータの参照時に発生したことがわかる．このストア命令に特化した対策を施すことでメモリ・システムを高性能化できる可能性がある．
- *181.mcf*: *179.art*, *188.ammp* と異なり，キャッシュ・ミスと深い関連のあるストア命令は発見できない．

以上から，ベンチマーク・プログラムの中には，ストアしたデータの多くが参照時にキャッシュ・ミスを発生させるような CS 命令が存在することが判明した．このような CS 命令が存在する場合には，それに特化した対策を施すことで性能向上が得られる可能性があると言える．したがって，CS 命令に着目することは，メモリ・システムの性能向上を考える上で妥当であると考える．

4. おわりに

本稿では，メモリ・システム高性能化手法開発において，DL 命令および CS 命令に着目することの妥当性について議論した．我々が行なった定量的な評価実験によれば，DL 命令によるキャッシュ・ミスがマイクロプロセッサ性能に与える影響は非常に大きく，DL 命令に着目することは妥当である．さらに，一部のベンチマーク・プログラムにおいては DL 命令によるキャッシュ・ミスの多くに関与している CS 命令が存在することが明らかになった．このことから， CS 命令に着目してメモリ・システムの性能向上を目指すことは理にかなっていると言える．

今後，我々はベンチマーク・プログラムのメモリ・アクセス・パタンを解析し，DL 命令がキャッシュ・ミスを頻発させる理由を解明する予定である．この際，プログラム中での処理と対応させながらその意味を検討する必要があると考えている．このようにして得られたキャッシュ・ミス原因を元に，DL 命令および CS 命令に着目したメモリ・システム高性能化技術を開発し，その効果を明らかにする．

文　献

[1] J. D. Collins, H. Wang, D. M. Tullsen, C. Hughes, G. Hoflener, D. Lavery and J. P. Shen: “Speculative precomputation: Long-range prefetching of delinquent loads”, Proc of the 28th Intl. Symposium on Computer Architecture (2001).

[2] A. Roth and G. Sohi: “Speculative data-driven multithreading”, Proc of the 7th Intl. Symposium on High-Performance Computer Architecture (2001).

[3] D. Kim and D. Yeung: “Design and evaluation of compiler algorithms for pre-execution”, Proc of the 10th Intl. Conference on Architectural Support for Programming Languages and Operating Systems (2002).

[4] S. S. Liao, P. H. Wang, H. Wang, G. Hoflener, D. Lavery and J. P. Shen: “Post-pass binary adaptation for software-based speculative precomputation”, Proc of the ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation (2002).

[5] C. K. Luk: “Tolerating memory latency through software-controlled pre-execution in simultaneous multithreading processor”, Proc of the 28th Intl. Symposium on Computer Architecture (2001).

[6] M. Annavaram, J. M. Patel and E. S. Davidson: “Data prefetching by dependence graph precomputation”, Proc of the 28th Intl. Symposium on Computer Architecture (2001).

[7] A. Moshovos, D. N. Pnevmatikatos and A. Baniasadi: “Slice-processors”, Proc of the Intl. Conference on Supercomputiong (2001).

[8] 三輪英樹, 堂後靖博, V. M. G. Ferreira, 井上弘士, 村上和彰：“キャッシュ・ミス頻発命令を考慮したメモリ・システムの高性能化”, 情報処理学会研究報告, 2004-ARC-160 (2004).

[9] D. Burger and T. M. Austin: “The simplescalar tool set, version 2.0”, University of Wisconsin-Madison Computer Sciences Department Technical Report (1997).

[10] J. Hennessy, N. Jouppi, S. Przybylski, C. Rowen, T. Gross, F. Baskett and J. Gill: “Mips: A microprocessor architecture”, Proceedings of the 15th annual workshop on Microprogramming (1982).

[11] R. L. Sites: “Alpha axp architecture”, Communications of the ACM, **36**, pp. 33–44 (1993).

[12] J. L. Henning: “Spec cpu2000: Measuring cpu performance in the new millennium”, **33**, 7 (2000).

[13] M. Oskin: “Pisa gcc 2.7.2.3 cross compiler”. http://arch.cs.ucdavis.edu/RAD/.

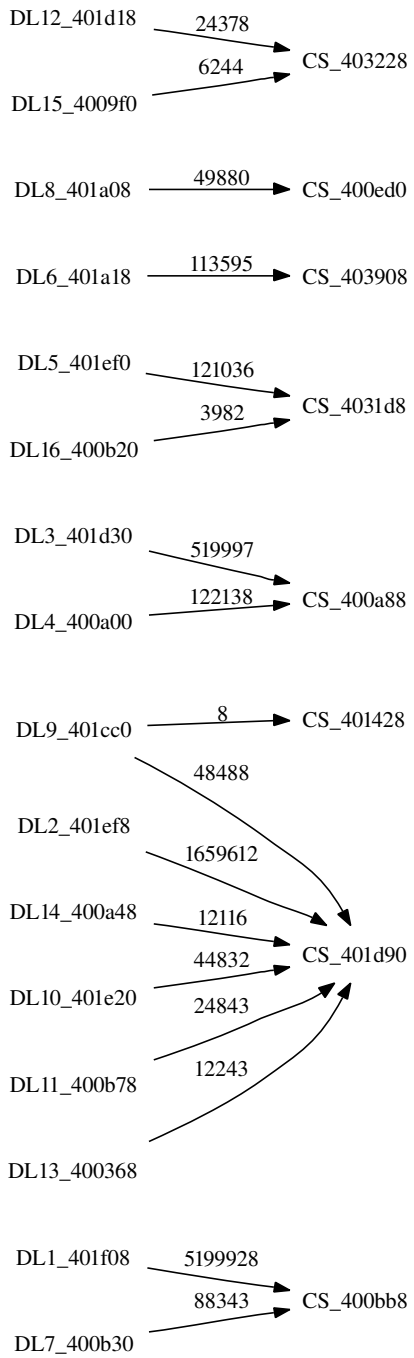

図 4　DL 命令と CS 命令の対応状況: *179.art*

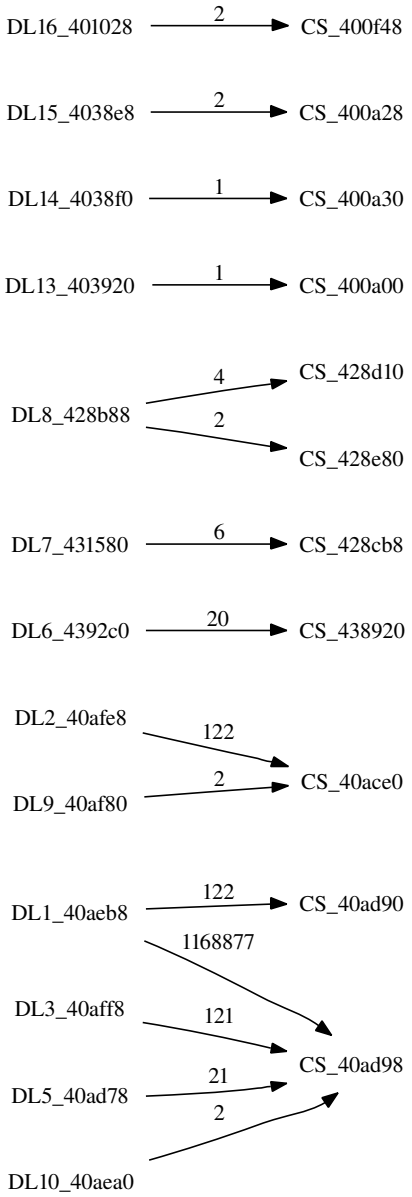

図 5　DL 命令と CS 命令の対応状況: *188.ammp*

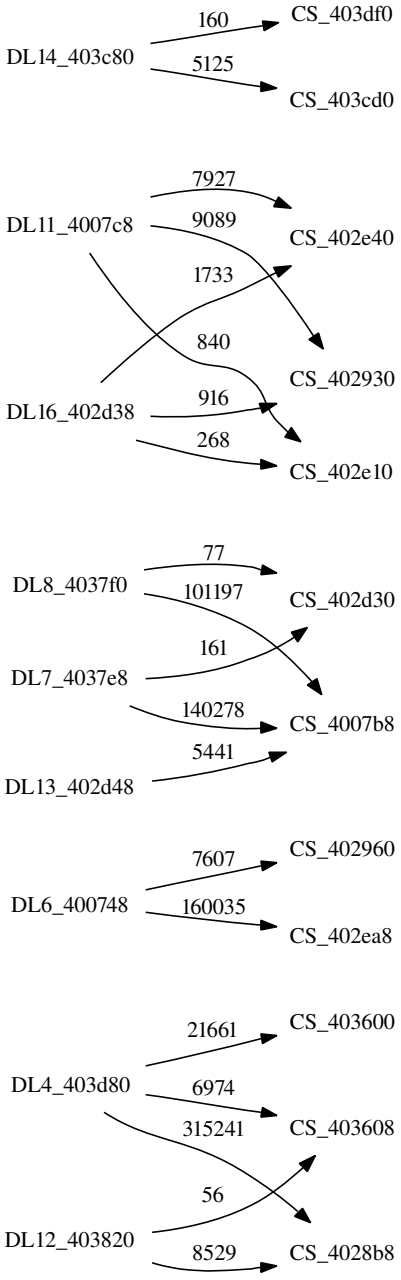

図 6　DL 命令と CS 命令の対応状況: *181.mcf*