

投機的手法を用いたデータ再利用による Java 仮想マシンの高速化

山田 克樹[†] 中島 康彦^{††} 富田 眞治[†]

Java 仮想マシンに対してデータ再利用を適用し、再利用表(RB)の挙動に関する調査を行った。RB ヒット率が 20.4%~76.0%とかなり高い数値を示すこと、また、再利用可否判定に用いる引数およびヒープ上データの個数が平均して各々3個未満と極めて少ないことが明らかになった。さらに、可否判定のためのデータ量が少ないことを利用して、投機的手法によりメソッドの引数を予測し、RBのエントリを投機的に用意する機構を提案する。本機構により、一般的なデータ投機において問題となるキャンセル時のペナルティを発生させることなく、投機の実行による高速化を図ることができると考えている。

Java Bytecode Execution Using Data Value Reuse with Speculative Preloading

KATSUKI YAMADA,[†] YASUHIKO NAKASHIMA^{††} and SHINJI TOMITA[†]

This paper describes the behavior of reuse buffer (RB) on Java virtual machine equipped with data value reuse. The RB hit ratio reaches 20.4%~76.0%. The average number of arguments and heap data for reuse test counts less than three respectively. We propose an RB with speculative preloading technique that predicts the arguments for each methods. This technique overcomes the penalties for cancellation that the general speculative technique essentially has.

1. はじめに

Java バイトコードの実行環境である Java 仮想マシン¹⁾(以下 JVM)の実装には、大きく以下の3つの方式がある。本稿は、ハードウェア直接実行方式に関するものである。

インタプリタ方式 ソフトウェアによりバイトコードを逐次解釈実行する方式である。実行に必要なメモリ量は少ないものの実行速度が遅い。

静的/動的命令変換方式 高速化のためにバイトコードをネイティブコードに変換してから実行する方式である。おおまかに、静的変換は全体を変換してから実行する方式、動的変換は必要に応じて部分的な変換を行いながら実行を進める方式である。最適化処理およびネイティブコードの格納のために十分なメモリを確保できる場合に極めて有効である。

ハードウェア直接実行方式 ハードウェアによりバイトコードを逐次解釈実行する方式である。高速性を追

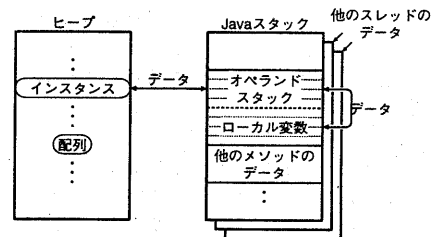


図1 JVMのデータ域

求しつつも、メモリ量や消費電力に対する制約がある場合に有効である。

JVM が使用するデータ域は図1のように大きく3つの部分から成る。

オペランド・スタック JVMの大半の命令は、オペランド・スタックから値を取得、演算し、結果を返す処理を行う。メソッドに引数を渡し、返り値を受け取るためにも使われる。

ローカル変数 メソッド内でのみ使用する値を保持している。メソッド呼び出しの引数はこの領域を用いて受け取る。

ヒープ クラス・インスタンスおよび配列の割り当て

[†] 京都大学大学院情報学研究科通信情報システム専攻
Division of Communications and Computer Engineering,
Graduate School of Informatics, Kyoto University
^{††} 京都大学大学院経済学研究科
Graduate School of Economics, Kyoto University

を行うためのデータ域である。

JVM では、全ての演算はスタックを経由する必要がある。このため、一般的な RISC プロセッサのように命令レベル並列性をバイトコード列から直接抽出することにより高速化を図ることは難しい。実行すべきバイトコードそのものを減らすために、データ投機およびデータ再利用の適用が考えられる。

2. 関連研究および本研究の背景

データ投機^{4),5)}およびデータ再利用^{6)~9)}に関しては近年多くの研究が行われている。しかし、バイトコードの実行に対して適用した研究成果は、ほとんど報告されていない。我々は、バイトコードの特徴である、1) 演算およびロード結果が必ずスタックトップに格納されること；2) 各命令に関連する記憶域がオペランド・スタック、ローカル変数、ヒープ領域のいずれであるかがオペコードにより容易に区別できること；に注目した。すなわち、命令実行結果が多くの汎用レジスタに格納され、また、オペコードだけではロード結果が局所変数であるか大域変数であるかの区別が難しい一般的な RISC プロセッサに比べ、データ投機およびデータ再利用を適用するための機構を単純化することができると考えた。

文献 10) では、SPEC JVM98²⁾の調査結果をもとに、有限個のレジスタ、有限容量のキャッシュを有する現実的な 5 段パイプライン構造を仮定し、データ投機およびデータ再利用の効果を測定した。予測表のエントリ数を無制限とした Last Value Prediction では、パイプラインハザードを生じる 33.0% から 47.2% (平均 38.5%) のバイトコードがデータ投機の対象となり、このうち予測が正しい命令は 24.3% から 66.9% (平均 54.8%)、全体の実行サイクル数に対する削減可能なサイクル数は 3.8% から 29.1% (平均 17.0%) であった。

この結果を見ると、単純なデータ投機の効果には魅力が感じられない。予測が正しくなかった場合のキャンセルおよび再実行のために、実際にはより多くのペナルティを要すること、また、複雑なハードウェアが必要となるためである。一方、データ再利用はキャンセルを行う必要がないことから、このようなペナルティが発生せず、今後有望であると考えている。さらに、入力と演算結果の両方を投機的に求め、再利用表 (Reuse Buffer, 以下 RB と略する) に対して正しいエントリを予め登録 (プリロード) する手法を確立することができれば、RB ヒット率を上げたり、より小さな RB でも高いヒット率を維持できる可能性がある。

本稿では、現状の RB の挙動を分析し、投機的手法により RB へのプリロードを行うための基本機構について考察する。

3. Java プロセッサの基本構成

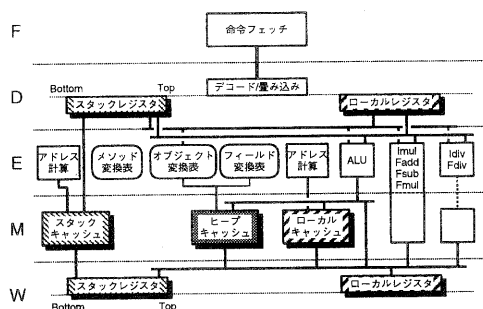


図2 Java プロセッサの構成

本稿において仮定する Java プロセッサの基本構成を図2に示し、概要を説明する。各パラメータは、Kaffe1.0b4³⁾上において、SPEC JVM98に含まれる compress, jess, db, javac, mpegaudio, mtrt を各々 s1, s10, s100 のサイズにより走行した合計 18 組の測定結果に基づくものである。ヒープ参照やメソッド呼び出し時のアドレス変換を高速化するために以下の変換表を用いる。

オブジェクト変換表 オブジェクト・リファレンスから、先頭アドレスおよび属性を得る。

フィールド変換表 コンスタント・プールへのインデックスから、フィールドのオフセットを得る。250 エントリ程度あれば、全変換対を収容できる。

メソッド変換表 コンスタント・プールへのインデックスから、先頭アドレスおよび属性を得る。300 エントリ程度あれば、ほぼ全変換対を収容できる。

ローカル変数上の値を直接参照するために、高速アクセスが可能なローカルレジスタを設け、ローカル変数間の演算をレジスタ間演算とする。ローカルレジスタから溢れたものは主記憶に保持する。主記憶上のローカル変数の値を使用して演算を行う場合には、これらの値を一度保持しておくレジスタが必要となる。この場合にはオペランド・スタックに対応するスタックレジスタを介して実行を行う。スタックレジスタの数は、多くとも1命令が1度に使うスタックのエントリ数だけあれば良いことから、8本とする。また、ローカル変数に関する主記憶アクセス高速化のためにローカルキャッシュを用意する。同様に、オペランド・スタックに関してスタックキャッシュを用意する。オペランド・スタックのトップに対するプッシュ/ポップに伴って必要となる、スタック・ボトムからキャッシュへのデータ退避、および、キャッシュからスタック・ボトムへのデータ供給は、必要に応じて自動的にプロセッサ内部で処理されると仮定した。

ローカルレジスタ上の値が更新される毎に、ローカルキャッシュへの書き戻しを行うと実行速度の低下を招く。これを避けるため、キャッシュへの書き戻しはメソッドの呼び出し時に行うこととした。またメソッドからのリターン時には、キャッシュに退避したレジ

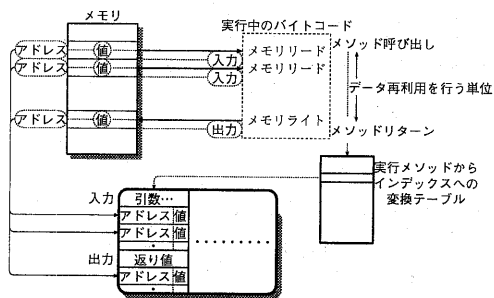


図3 RBの構造

表1 RBの諸元

RBの総エントリ数	32768
同時に比較を行うエントリ数の上限	128
引数比較の上限	8
配列比較の上限	4
配列格納の上限	2
フィールド比較の上限	4
フィールド格納の上限	2
スタティック比較の上限	2
スタティック格納の上限	1

スタの値を復元する。この他、実行するメソッドを切り替える時には、引数および戻り値の受け渡しを行う必要がある。この操作には、スタックレジスタとローカルレジスタの間でのデータの移動が必要となる。また、メソッド呼び出し、リターン時には、プロセッサの内部情報の退避、復元を行う必要がある。メソッド呼び出し、リターン時に必要な内部状態レジスタの退避および復元には10サイクル、また、退避および復元に必要となるローカル変数の個数は8とし、1サイクルにより1個の退避および復元が可能であるとする。これらを合計すると、1回のメソッド呼び出しおよびリターンに要するサイクル数はそれぞれ18となる。

4. データ再利用の概要

本稿では、メソッドを単位とするデータ再利用を行っている。過去のメソッド呼び出しに伴う全読み出しデータを登録しておき、参照したアドレスおよび値が以前と同じである場合に、再利用が可能であると判断し、登録してある書き込みデータを書き戻す。図3および表1にRBの構造および諸元を示す。RBの各エントリは、過去にメソッドに渡された引数、メソッドが読み出したヒープ上データ（配列、フィールド、スタティック）の各アドレスおよび内容、書き込んだヒープ上データの各アドレスおよび内容、戻り値を保持している。動作を以下に示す。

- (1) メソッド呼び出しに対し、引数から比較の候補とするRBのインデックスを最大128個求める。
- (2) RBに登録されている過去の引数およびヒープからの読み出しデータと、現在の引数およびヒープ上

データを比較する。ヒープとの比較には1アドレスにつき1サイクルを要すると仮定する。

(3) 再利用が可能である場合には、登録してある書き込みデータを書き戻し、戻り値を次命令へフォワードする。

(4) 再利用が不可能である場合には、実際にメソッドを呼び出し、図1に納まる範囲の実行結果をRBに登録していく。メソッドが入れ子になっている場合、下位のメソッドが参照したヒープに関する情報を上位のメソッドに対応するRBにも登録する。これにより上位のメソッド全体の実行を省略することもできる。

ただし、ネイティブメソッドが呼び出された場合、登録中のメソッド全てについてRBへの登録を取り消している。これは、ネイティブメソッドにおいて出力等が発生する可能性があり、正しいデータ再利用を保証することができないためである。

5. 再利用の効果

以上のようなデータ再利用を適用した結果、表2に示すように、全体の実行サイクル数に対する削減可能なサイクル数は1.09%から47.0%（平均16.7%）であった。文献10)では引数比較およびフィールド比較の上限を各々16としていたのに対し、表1では各々8と4に縮小した。若干数値が異なっているものの、エントリ数の縮小がほとんど影響を与えないことがわかる。

次に、各メソッド呼び出しに注目したRBの挙動を表3および表4に示す。RB参照回数は、メソッド呼び出しが図1の条件を満たしRBに最低1エントリが登録されたためにRBの参照が行われた回数、RB参照率は全メソッド呼び出し回数に対するRB参照回数の割合である。RBヒット回数は、再利用が可能であった回数、RBヒット率はRB参照回数に対するRBヒット回数の割合である。また、引数比較平均は、ヒット時の引数の平均個数、ヒープ比較平均は、同じくヒープ上データの平均個数である。

RBヒット率が20.4%~76.0%とかなり高い数値を示していることは興味深い。また、RBに登録可能な引数およびヒープ上データが最大8個および10(4+4+2)個であるのに対し、実際に必要となるのは各々平均3個未満と極めて少ないことがわかる。

6. メソッド毎のRBヒット率の分析

RBヒット率が低いメソッド呼び出しに先立ち、引数を予測し、正しい実行結果を予めRBに登録することができれば、RBヒット率を上げたり、より小さなRBでもヒット率を維持できる可能性がある。比較の対象となる引数の個数が比較的少ないことが判明したために、引数を予測することが現実味を帯びてくる。

次の段階として、メソッドの引数を予測する余地が

表2 再利用により削減可能なサイクル数の割合 (単位は%)

サイズ	compress	jess	db	javac	mpegaudio	mtrt
s1	11.4	26.5	11.4	4.68	10.5	47.0
s10	7.98	44.9	13.3	15.1	7.60	1.09

表3 メソッド呼び出しに注目した分析 (s1)

	compress	jess	db	javac	mpegaudio	mtrt
メソッド呼び出し	17359045	579805	105180	629304	1136453	6044166
RB 参照回数	15538113	355057	52220	181516	842789	4608397
RB 参照率 (%)	89.5	61.2	49.6	28.8	74.2	76.2
RB ヒット回数	4194462	192576	13920	37036	401213	3500838
RB ヒット率 (%)	27.0	54.2	26.7	20.4	47.6	76.0
引数比較平均 (個)	2.00	1.33	1.21	0.91	1.57	1.13
ヒープ比較平均 (個)	2.00	1.41	1.57	0.79	0.86	1.13

表4 メソッド呼び出しに注目した分析 (s10)

	compress	jess	db	javac	mpegaudio	mtrt
メソッド呼び出し	18198757	6024455	1867579	4493950	9300517	24391637
RB 参照回数	15582153	4964854	1119574	1959292	7273108	862944
RB 参照率 (%)	85.6	82.4	59.9	43.6	78.2	3.54
RB ヒット回数	3331209	3511692	464445	912528	2892786	317969
RB ヒット率 (%)	21.4	70.7	41.5	46.6	39.8	36.8
引数比較平均 (個)	2.00	1.91	1.58	1.13	1.59	0.77
ヒープ比較平均 (個)	2.00	1.16	2.32	0.75	0.85	0.80

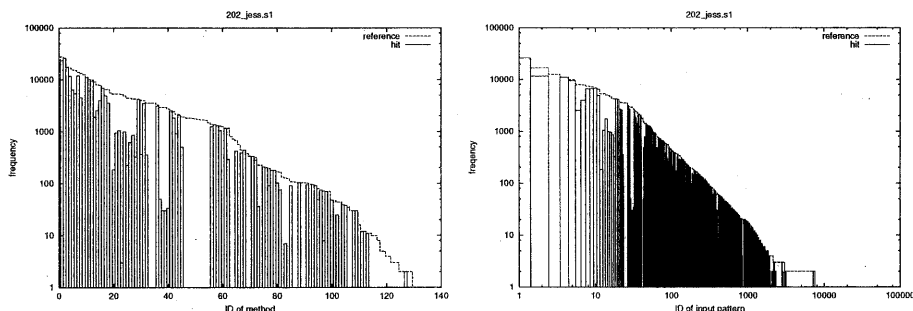


図4 jess(s1) のメソッド別、引数パターン別 RB 参照状況

あるかどうかについて調査した。出現頻度が極めて高い一方、RB ヒット率が極めて低いメソッドが引数予測の候補として適当である。このようなメソッドを探索するために、jess と db を対象として、メソッド毎の RB 参照回数とヒット回数、および、メソッドに引数を加えた比較パターン毎の RB 参照回数とヒット回数を図4、図5、図6、図7に示す。

各図の左側のグラフは、RB 参照回数の多い順に、X 軸方向のメソッドを並べ替え、RB 参照回数およびヒット回数を対数表示したものである。また右側のグラフは、メソッドに引数を加えた比較パターンを X 軸方向とし、X 軸についても対数表示したものである。

図4と図5を比較すると、jess(s1) に比べて jess(s10) はわずかに数個のメソッドの RB 参照回数および RB ヒット率が極めて高いために、全体の RB ヒット率を押し

上げていることがわかる。jess(s10) の比較パターンを見ると、RB 参照回数の多いパターンのほとんどが RB にヒットしている。これに対して jess(s1) では、上位のパターンにヒット率が低いものが比較的多く含まれている。さらに調査した結果、jess(s1) においてヒット率の低いパターンは、jess 自身の入力データに直接関係するものであることがわかった。問題サイズが小さいために入力に関する処理の比率がかえって高くなると考えられる。

この傾向は図6および図7の比較においてより顕著である。db(s1) では入力データに直接関与するメソッドが上位の大部分を占めるために、RB ヒット率が低い。db(s10) では RB ヒット率の高いメソッドが上位を占めるために、全体の RB ヒット率が高くなっている。グラフを記載していない javac についても同様の

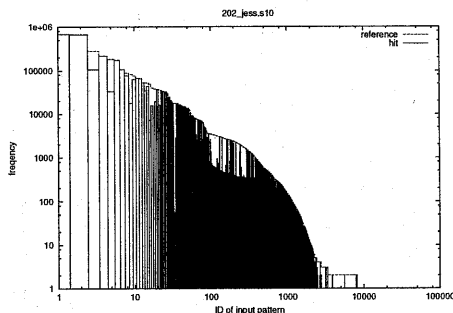
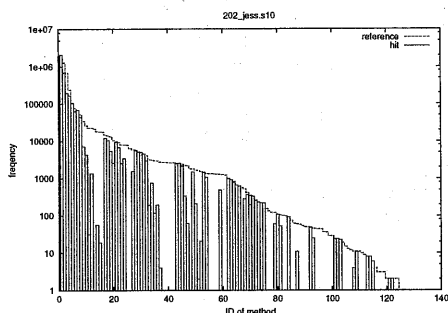


図5 jess(s10)のメソッド別、引数パターン別RB参照状況

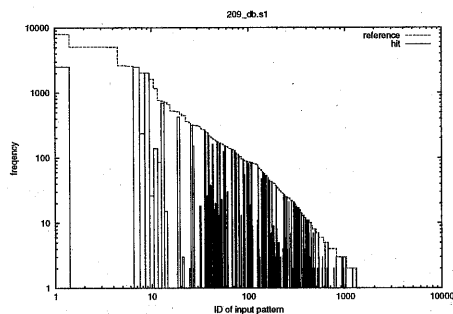
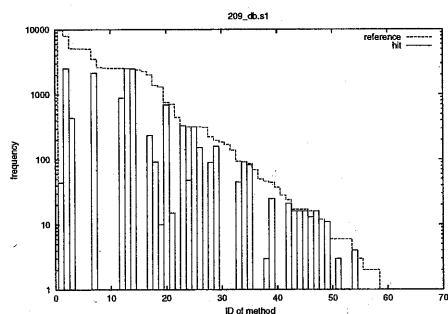


図6 db(s1)のメソッド別、引数パターン別RB参照状況

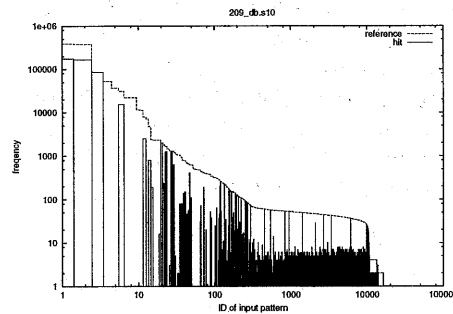
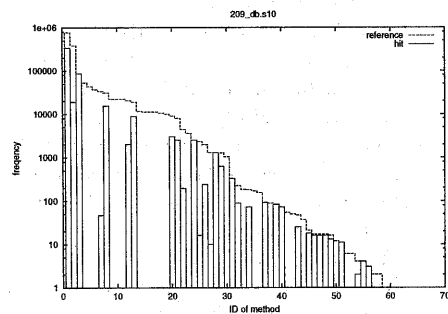


図7 db(s10)のメソッド別、引数パターン別RB参照状況

調査結果が得られている。

現在の調査範囲では、出現頻度が極めて高い一方、RB ヒット率が極めて低いようなメソッドのほとんどは、残念ながら、プログラムの入力に直接影響を受ける引数を用いている。従って、現時点では投機的実行によりRB ヒット率を格段に向上させることは難しい。他のベンチマークプログラムについても詳細に調査する必要がある。

7. 投機的手法によるRBへのプリロード

jessの実行の際、出現頻度がそれほど高くないもの

の、引数がほぼ単調に増加するメソッドがKaffe1.0b4内部のライブラリ中に見つかっている。また、mtrtにおいて出現頻度が高いメソッドのいくつかは、ヒープ上の同一フィールドを連続して数回参照し、次のフィールド位置へ移動することを繰り返している。引数が単調に変化する場合には、過去の演算結果が再利用されることはなく、変化直後のRB ヒット率は0である。一方、単調変化を予測して引数および実行結果をRBに登録しておくことができれば、ヒット率をさらに高めることができる。図8に、このような投機的手法を適用するためのハードウェア構成を例示する。(A)は、アドレス予測によるキャッシュへのプリロードとの類

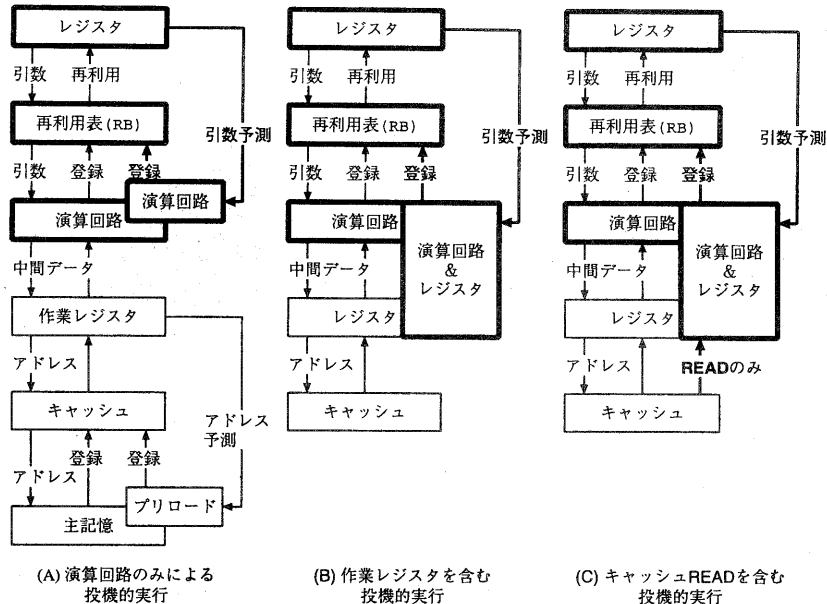


図8 RBへのプリロード機構

似性を示した図である。多くのサイクル数を要する単一演算においてソースデータに局所性が存在する場合には、通常の演算回路とは別に、予測したソースデータに対して演算を行う演算回路を用意すればよい。単一演算では不十分である場合には、(B)のように作業レジスタを追加することが考えられる。さらに、ヒープの内容も必要とする場合には、(C)のようにキャッシュからの読み出しパスの追加が必要となる。投機的演算結果をキャッシュではなくRBにのみ書き込むことにより、引数の予測が外れた場合でも投機的実行のキャンセルを不要とすることができる。

8. ま と め

本稿では、Java 仮想マシンに対してデータ再利用を適用し、再利用表(RB)の挙動を調査した。再利用可否の判定のためのデータ量が少ないことから、さらに、引数および実行結果を投機的手法により求めるデータ再利用の可能性について考察を行った。今後は、引数を効率良く予測するための具体的な方法、および、図8の(A)から(C)の各構成におけるコストと効果について、定量的な評価を進める予定である。

参 考 文 献

- 1) リンドホルム, T., イェリン, F. (野崎訳): The Java 仮想マシン仕様, アジソン・ウェスレイ・パブリッシャーズ・ジャパン (1997).
- 2) SPEC JVM98 VERSION 1.03, the Standard Performance Evaluation Corporation (1998).
- 3) Kaffe.org: Welcome to Kaffe,

- <http://www.kaffe.org/>
- 4) Lipasti, M.H., Shen, J.P.: Exceeding the Dataflow Limit via Value Prediction, 29th International Symposium on Microarchitecture, pp.226-237 (1996).
 - 5) Wang, K., Franklin, M.: Highly Accurate Data Value Prediction using Hybrid Predictors, 30th Annual International Symposium on Microarchitecture (1997).
 - 6) Sodani, A., Sohi, G.S.: Understanding the Differences Between Value Prediction and Instruction Reuse, 31st Annual ACM/IEEE International Symposium on Microarchitecture (1998).
 - 7) Sodani, A., Sohi, G.S.: Dynamic Instruction Reuse, 24th Annual International Symposium on Computer Architecture, pp.194-205 (1997).
 - 8) González, A., Tubella, J., Molina, C.: Trace-Level Reuse, 1999 International Conference on Parallel Processing (1999).
 - 9) Huang, J., Lilja, D.J.: Exploiting Basic Block Value Locality with Block Reuse, The Fifth International Symposium on High Performance Computer Architecture (1999).
 - 10) 重田大助, 小川洋平, 山田克樹, 中島康彦, 富田眞治: 命令量込み込み, データ投機および再利用技術を用いた Java 仮想マシンの高速化, 情報処理学会研究報告, ARC-137-3, pp. 13-18 (2000).