計算機アーキテクチャ　112－4
（１９９５．　６．　２３）

ロード命令の先行実行の科学技術計算プログラムによる評価

小沢年弘[†]　　西崎慎一郎[††]　　木村康則[†]

[†](株) 富士通研究所　　[††](株) 富士通ソーシアルサイエンスラボラトリ

非数値計算プログラムのメモリ系のオーバーヘッドを低減するために、我々は、これまでにヒューリスティックスを利用したロード命令の先行実行を行なう方法を開発した。今回、この方法を科学技術計算プログラムにも適用できるように拡張を施した。この方法では、一定のストライドか、不定値でアドレスが変化するループ中のロード命令はキャッシュ・ミスを起こし易いというヒューリスティックスを新たに使用して、先行実行するロード命令を選択している。

このヒューリスティックスにより、科学技術計算プログラムを含めて、キャッシュ・ミスを起こし易いロード命令の大部分を選択できることを示した。また、この方法により、科学技術計算プログラムにおいて5～27％の実行時間短縮が得られ、大きなキャッシュ・ミス・レイテンシィのシステムにおいても、ロードの先行実行の効果が得られることを示した。

Evaluation of Preload Technique for Scientific Programs

ToshihiroOzawa[†], *Shin'ichiroNishizaki*[††], *YasunoriKimura*[†]

[†]FUJITSU Laboratories LTD. [††]Fujitsu Social Science Laboratory Ltd.

1015. Kamikodanaka Nakahara-ku, Kawasaki 211, Japan

email: ozawa@flab.fujitsu.co.jp

We had developed a preload method for non-numerical programs. It used heuristics to identify which load instructions tend to cause a cache-miss on non-numerical programs. We have improved this method to apply to numerical programs. We use a new heuristic that load instruction in a loop with a constant or variable address increment tends to cause a cache-miss.

We show that most of load instructions which tend to cause a cache-misss are selected by the heuristics for any type of programs. We also show that the reduction in the execution time for numerical programs ranges from 5% to 27% and that the method is effective for numerical programs with long cache miss latency.

1 はじめに

近年、プログラム実行時にプロセッサへデータを如何にうまく供給するかという問題が重要になっている [1]。これまで計算機システムでは、記憶構成を階層的にし、遅い記憶装置へのアクセスをより速い記憶装置へのアクセスに変換することにより、プロセッサへのスムーズなデータの供給を行なわせてきた。しかし、従来、キャッシュ・ミス時には、データをメモリからキャッシュに持ってくるまでの時間（キャッシュ・レイテンシィ）、プロセッサが止まってしまっていた（プロセッサ・ストール）。プロセッサの性能が急速に向上していることから、キャッシュ・ミスによるプロセッサ・ストール時間が相対的に大きくなっており、この低減が求められている [2][3]。

この問題を解決する一手法として、ロード命令の先行実行の方法が研究されてきた [4][5]。この方法では、non-blocking ロード命令[1]を備えたハードウエアを用意し、キャッシュ・ミス時に、ロード命令の後続命令をメモリ・アクセスと並列に実行し、メモリ・ミス時のオーバーヘッドを隠蔽する方法である。この方法の効果は、キャッシュ・ミスを起こしたロード命令と、そのロードしたデータを使用する命令（使用命令と呼ぶ）の間に、どのくらい多くの命令を実行させられるかによって決まる。従って、この方法の効果を上げるには、コンパイラによって、キャッシュ・ミスを起こし易いロード命令とその使用命令との間に多くの命令をスケジューリングしたコードを生成することが必要になる。そのために、キャッシュ・ミスを起こし易いロード命令をコンパイル時に選択する技術、および選択したロード命令とその使用命令の間に多くの命令をスケジューリングする技術の二つが重要になる。

我々は、これまでに非数値計算プログラムに対して、キャッシュ・ミスを起こし易いロード命令をコンパイラにより見つけるためのヒューリスティックス（キャッシュ・ミス予測のヒューリスティックス）と、キャッシュ・ミスのオーバーヘッドを低減させる命令スケジューリング法について報告した [6]。本稿では、これらの技術を科学技術計算プログラムに対しても適用できるように改良した方法と、その評価について、非数値計算プログラムと対比させて述べる。

[1]ロード命令がキャッシュ・ミスを起こした時、その完了を待たずに後続命令の実行が行なえるロード命令。ただし、後続命令が、ロード命令がロードしてくる値を使用する場合には、その命令の実行は、キャッシュ処理の完了まで待たされる。

1.1 先行実行すべきロード命令の選択

1.1.1 キャッシュ・ミス予測のヒューリスティックス

キャッシュ・ミス予測のヒューリスティックスとして、次のような基準を得ることが目標となる。つまり、キャッシュ・ミスの大部分が、その基準に適合するロード命令で起きており、かつその基準に適合するロード命令が、より少ないような基準を見つけることである。

以下のような基準に合うロード命令がキャッシュ・ミスを起こし易いというのが、非数値計算プログラムに対するキャッシュ・ミス予測のヒューリスティックスであった。

- **list access:** ロードしたデータをベース・アドレスとして、さらにロードを行なう場合
- **big stride access:** ループ内で、キャッシュ・ブロックを越える stride でデータを読み出す可能性のある場合

しかし、このヒューリスティックスを科学技術計算プログラムに適応しても、キャッシュ・ミスを起こし易いロード命令を適切に選別することはできなかった。科学技術計算プログラムでは、メモリ参照は、配列への規則的な参照により行なわれることが多い。stride が小さい場合でも、このような参照におけるキャッシュ・ミス率は大変大きなものとなる。**big stride access** のヒューリスティックスは、このような小さい stride の規則的な参照をカバーしていない。そこで、科学技術計算プログラムのキャッシュ・ミスも予測できるように、**big stride access** の代わりに、以下のヒューリスティックスを導入した。

表 1: 非数値計算プログラムにおけるロード命令の種類別の静的割合（%）

program	list access	stride access	合計
008.espresso	16.3	16.5	32.8
022.li	19.8	10.3	30.1
023.eqntott	20.4	13.9	34.3
026.compress	7.0	12.8	19.8
072.sc	13.1	20.6	33.7
085.gcc	28.3	11.3	39.6

表 2: 科学技術計算プログラムにおけるロード命令の種類別の静的割合（%）

program	list access	stride access	合計
047.tomcatv	0.0	65.0	65.0
052.alvinn	0.0	44.4	44.4
056.ear	4.3	10.0	14.3
078.swm256	5.2	32.1	37.3
090.hydro2d	3.8	34.8	38.6
093.nasa7	3.5	54.0	57.5

- **stride access:**ループ内で、1ワード以上のstrideでデータを読み出す可能性のある場合

stride access は、**big stride access** を含むことになる。

1.1.2 キャッシュ・ミス予測のヒューリスティックスの評価

改良したキャッシュ・ミス予測のヒューリスティックスの評価を非数値計算プログラムと科学技術計算プログラムに対して行なった。対象とするプログラムは、非数値計算プログラムとしては、`Specint92` の全プログラム、科学技術計算プログラムとしては、`Specfp92` の内から6つを選んだ。

静的な評価として、コンパイル時に制御の流れを追い、ロード・アドレスを解析することによって、各ロード命令を、**list access**、**stride access**、その他に分類した。ただし、ループ中に含まれるロード命令で、そのロード・アドレスを特定できない場合には、**stride access** に分類している。

非数値計算プログラムと科学技術計算プログラムそれぞれについて、ロード命令の種類別の静的な割合を、表1、表2に示す。

また、動的なデータとして、表3に示した入力データを用い、シミュレータ上で各ベンチマーク・プログラムを実行し、ロード命令によるキャッシュ・ミス回数を、ロード命令の種類別に集計した。

表 3: ベンチマーク・プログラムへの入力

program	入力
008.espresso	input.ref/bca.in
022.li	input.veryshort/li-input.lsp.8
023.eqntott	input.ref/int_pri_3.eqn
026.compress	input.ref/in
072.sc	input.ref/loada1
085.gcc	input.ref/1cexp.i
047.tomcatv	-
052.alvinn	-
056.ear	input.ref/ref.m22
078.swm256	input.ref/swm256.in
090.hydro2d	input.ref/hydro2d.in
093.nasa7	-

データ・キャッシュの構成が、

- `16K1w`: cache size: 16Kbytes, block size: 32bytes, direct mapped cache
- `16K2w`: cache size: 16Kbytes, block size: 32bytes, 2way set associative cache
- `32K1w`: cache size: 32Kbytes, block size: 32bytes, direct mapped cache
- `32K2w`: cache size: 32Kbytes, block size: 32bytes, 2way set associative cache

の場合についての結果を、図1、図2に示す。ただし、命令キャッシュは、ミスをしないものとした。

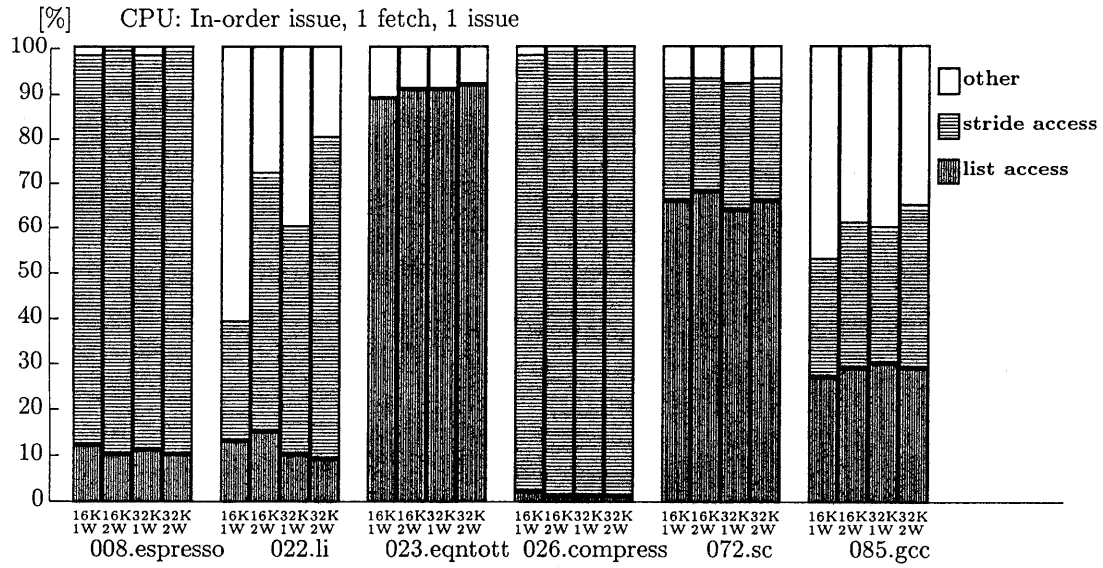

図 1: 非数値計算プログラムにおけるロード命令の種類別キャッシュ・ミス回数の割合

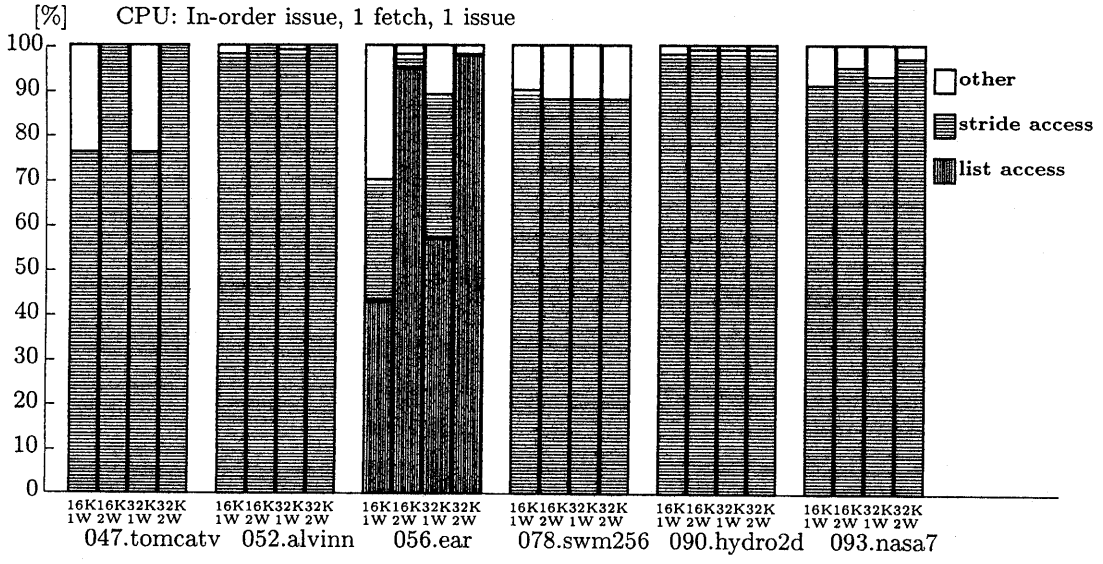

図 2: 科学技術計算プログラムにおけるロード命令の種類別キャッシュ・ミス回数の割合

非数値計算プログラムでは、ヒューリスティックスに適合するロード命令によって起こされるキャッシュ・ミス回数は、高い割合を示している。しかし、ヒューリスティックスとして **stride access** を使用した場合と比べて、**big stride access** を使用しても、ほとんど変化していない。一方、ヒューリスティックスに適合するロード命令の静的な割合は、**stride access** を使用することによって、**big stride access** を使用するよりも、８％程度増加する。これは、非数値計算プログラムに関しては、**stride access** よりも、**big stride access** の方が、ヒューリスティックスとして優れていることを示している。

これに対して、科学技術計算プログラムでは、ヒューリスティックスに適合するロード命令によって起こされるキャッシュ・ミス回数の割合を高くするためには、**stride access** をヒューリスティックスとすることが必須であった。ヒューリスティックスに適合するロード命令の静的な割合は、非数値計算プログラムに比べて、かなり大きくなっている。これは、科学技術計算プログラムの基本的な構造が、ループ構造による配列要素への規則的参照であることを示している。

stride access を使用したヒューリスティックスが、プログラムの種類によらないヒューリスティックスとなっている。このヒューリスティックスによれば、ほとんどのプログラムにおいて、全キャッシュ・ミスの８０％以上の原因となるロード命令を選択できる。選択されたロード命令は、全ロード命令の１５～６５％程度であった。一部の科学技術計算プログラムでは、半分以上のロード命令が選択されてしまっているが、大部分のキャッシュ・ミス・をカバーするという、ヒューリスティックスの目的は達成している。

キャッシュ・ミスの原因となるロード命令を、より的確に選択することが理想である。このためには、プログラムが処理する配列要素の大きさの識別や、unrolling により、ループの繰り返しを複製し、その中の一部のロード命令を選択することなどが有効であるかもしれない。

1.2 メモリ系オーバーヘッドを低減させる命令スケジューリング

我々は、先の報告 [6] で、キャッシュ・ミスのオーバーヘッドを隠蔽するための命令スケジュール方法について述べた。つまり、

- キャッシュ・ミスを起こすと予測されたロード命令を、なるべく早くスケジューリングする。
- その使用命令を、なるべく遅くスケジューリングする。
- キャッシュ・ミスを起こすと予測されたロード命令と、その使用命令の間に他の命令を移動する。
- これらのスケジューリング、移動に際して、投機的移動はループから脱出する分岐命令を越える場合以外は、行なわない。

科学技術計算プログラムに対して使用した命令スケジュール方法も同じものである。

この方法では、ベーシック・ブロック間に渡って、命令の移動が行なわれる。科学技術計算プログラムでは、非数値計算プログラムに比べて、一つのベーシック・ブロックに含まれる命令数が大きい傾向があり、ベーシック・ブロック内だけの移動で、キャッシュ・ミスを起こすと予測されたロード命令とその使用命令を十分離してスケジューリングできる場合も多かった。また、ループの先頭近くにあるロード命令を、ループの外とループの最後に複写／移動するスケジューリングが、科学技術計算プログラムでは多く現れた。これは、ループの各繰り返しの先頭の処理の一部を、一つ前の繰り返しにおいて実行することになり、キャッシュ・ミス以外の原因によるプロセッサ・パイプライン・ストールを減らすための効果も期待できる [7]。

2 ロード命令先行実行の評価

前章のようなスケジューリングを行なうスケジューラを GNU の SPARC 用 C コンパイラに組み込んだ。このコンパイラを Preload コンパイラと呼ぶ。

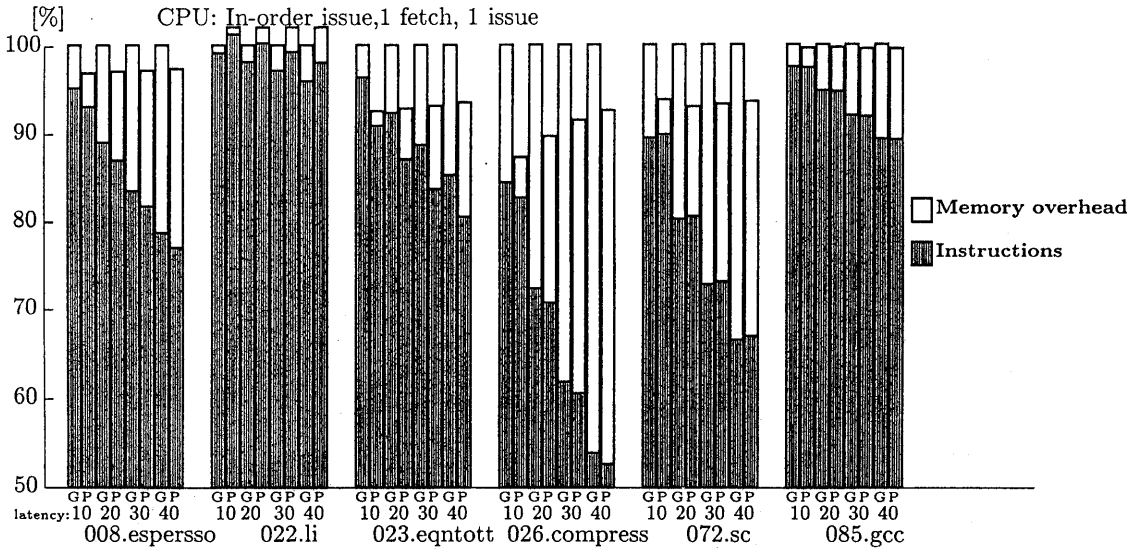

図 3: GCC コンパイラと Preload コンパイラのプログラム実行時間の比（非数値計算プログラム）

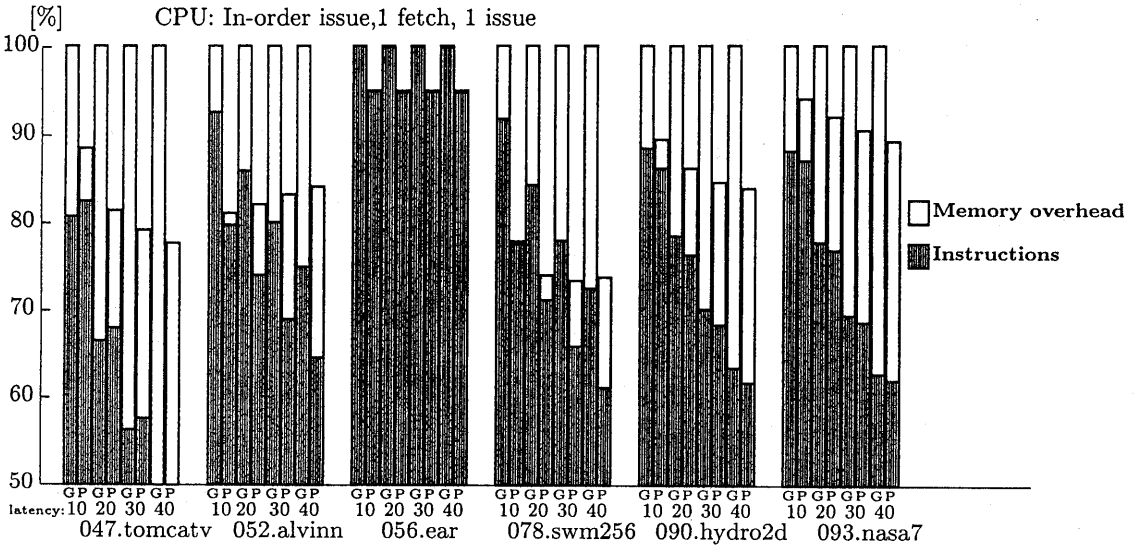

図 4: GCC コンパイラと Preload コンパイラのプログラム実行時間の比（科学技術計算プログラム）

各ベンチマーク・プログラムに対して、オリジナルのGNU Cコンパイラで生成したコードと、Preloadコンパイラで生成したコードの実行時間を、シミュレータにより求めた[8]。図3、図4に、非数値計算プログラム、科学技術計算プログラムの結果を示す。キャッシュ・ミスのレイテンシィが、10, 20, 30, 40サイクルの場合について求めている。図において、GはGNU Cコンパイラのコードの実行時間、PはPreloadコンパイラのコードの実行時間を示す。ただし、各実行時間は、そのキャッシュ・ミス・レイテンシィにおけるGNU Cコンパイラのコードの実行時間により正規化してある。また、各実行時間は、命令を実行するために使用した時間(Instructions)と、キャッシュ・ミスによるストールの時間(Memory overhead)に分けて表してある。

科学技術計算プログラムと、非数値計算プログラムにおけるGCCコンパイラのMemory overheadの大きさの比較から、科学技術計算プログラムは、非数値計算プログラムに比べて、キャッシュ・ミスのオーバーヘッドが大きいことが分かる。Preloadコンパイラを使用することによるキャッシュ・ミスのオーバーヘッドの減少量は、科学技術計算プログラムの方が大きく出ており、Preloadコンパイラを使用したことによる実行時間短縮も、科学技術計算プログラムでは、5〜27 %と大きくなっている。また、非数値計算プログラムでは、キャッシュ・ミス・レイテンシィが大きくなるも、Preloadコンパイラの実行時間短縮は大きくなっていないが、科学技術計算プログラムでは、レイテンシィの増加に伴って、Preloadコンパイラの実行時間短縮が大きくなる傾向が見られている。

これらは、非数値計算プログラムに比べて、

- 科学技術計算プログラムは構造が単純であり、ベーシック・ブロック間に渡った命令移動を行ない易いこと
- ベーシック・ブロックの大きさが比較的大きいこと
- 浮動小数点計算などのように複数サイクルかかる命令が多く使用されていること

から、キャッシュ・ミスを起こすと予測されたロード命令と、その使用命令の間に多くの命令をスケジューリングすることができ、かつロード命令の発行から、使用命令の発行までのサイクル数が、命令数以上に大きくなり、ロード命令の先行実行の効果を大きく出せているためである。したがって、キャッシュ・ミスのレイテンシィが40サイクルとかなり大きくなっても、Preloadコンパイラの効果が大きくなる場合が得られている。

3 まとめ

科学技術計算プログラム、非数値計算プログラムの両方に対して、キャッシュ・ミスを静的に予測するヒューリスティックスとして、

- **list access:** ロードしたデータをベース・アドレスとして、さらにロードを行なう場合
- **stride access:** ループ内で、1ワード以上のstrideでデータを読み出す可能性のある場合

を見い出した。大部分のプログラムにおいて、ロード命令によるキャッシュ・ミスの８０%以上が、ヒューリスティックスに当てはまるロード命令で起きていることを示した。静的な割合では、ロード命令の15〜65 %が、このヒューリスティックスに当てはまり、科学技術計算プログラムでは、割合が高くなってしまっている。

ロード命令の先行実行を行なうコンパイラ(Preloadコンパイラ)により、科学技術計算プログラムをコンパイルし、キャッシュ・ミスのオーバーヘッド削減の効果を評価した。

その結果、

- 非数値計算プログラムに比べて、科学技術計算プログラムでは、ロード命令の先行実行の効果が、大きく得られ、5〜27 %の実行時間の短縮が得られた。
- 科学技術計算プログラムでは、キャッシュ・ミスのレイテンシィが40サイクルとかなり大きくなっても、Preloadコンパ

イラの効果が大きくなる場合が得られている。

参考文献

[1] A. Gupta, J. Hennessy, K. Bharachorloo, T. Mowry, and W-D. Webber. Comparative Evaluation of Latency Reducing and Tolerating Techniques. In Proceedings of the 18th Annual Int. Symp. on Computer Architecture, 1991.

[2] J. D. Gee, M. D. Kill, D. N. Pnevmatikatos, and A. J. Smith. Cache Performance of the SPEC92 Benchmark Suite. In IEEE MICRO, Vol. 13, No. 14, Aug., 1993.

[3] A. R. Lebeck, and D. A. Wood. Cache Profiling and the SPEC Benchmarks: A Case Study. In IEEE Computer, Oct., 1994.

[4] T.-F. Chen, and J.-L. Baer. Reducing Memory Latency via Non-blocking and Prefetching Caches. In Proceedings of 5th ASPLOS, 1992.

[5] W. Y. Chen, S. A. Mahlke, and W. W. Hwu. Tolerating First Level Memory Access Latency in High-Performance Systems. In Int. Conf. on Parallel Processing, 1992.

[6] 小沢、西崎、木村. ロード命令の先行実行とその評価. 情報処理学会研究会報告 94-ARC-109-1, 1994.

[7] M. S. Lam. Software Pilelining: An Effective Scheduling Technique for VLIW Machine. In Proceedings of SIGPLAN Conference on Programing Language Design and Implementation, ACM. 1988.

[8] 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, 1993.