計算機アーキテクチャ 139－12
（2000．8．4）

線形リストを対象としたデータプリロード方式の評価

山村 周史†, 平田 博章†, 新實 治男‡, 柴山 潔†

† 京都工芸繊維大学 工芸学部 電子情報工学科
〒606-8585 京都市左京区松ヶ崎御所街道町
‡ 京都産業大学 工学部 情報通信工学科
〒603-8555 京都市北区上賀茂本山

線形リスト構造に対して，単純なハードウェア機構を用いて効果的にデータのプリフェッチ/プリロードを行う手法を提案している．本論文では，ポインタ操作が頻出するベンチマークプログラムを用いて，本プリロード方式の詳細な評価を行った．評価の結果，プロセッサ内部にデータをプリロードすることの効果が大きいことを確認した．また，投機実行において，誤ったプリロード値の使用は発生せず，性能低下の要因とはならないことも確認した．

An Evaluation of A Data Preload Mechanism for A Linked List Structure

Shuji Yamamura†, Hiroaki Hirata†, Haruo Niimi‡, Kiyoshi Shibayama†

† Dept. of Electronics and Information Science, Kyoto Institute of Technology
Matsugasaki, Sakyo-ku, Kyoto 606-8585 JAPAN
‡ Dept. of Information and Communication Sciences, Kyoto Sangyo University
Motoyama, Kamigamo, Kita-ku, Kyoto, 603-8555 JAPAN

We had proposed an efficient data preloading/prefetch mechanism for linked list structures. In this paper, we have evaluated our preload scheme by using a benchmark suite which includes many pointer operations. Our simulation results show that the preloading of link pointers can achieve greater processor performance improvement than only prefetching employed. We have not encountered the case of using an incorrectly preloaded value, and so, the speculation failure does not damage the performance in actual program execution.

1 はじめに

近年，データ値予測やデータアドレス予測，データプリフェッチといった，メモリへのアクセス遅延を隠蔽するための技術が盛んに研究されている．しかし，現在までに提案されている大部分の方式は，配列のような規則的なアドレスに配置されたデータのみを対象としている．また，連結リスト構造を対象とした方式も提案はされているが，いずれもプリフェッチを行うために大規模なハードウェアサポート機構を必要とする．

これに対し，我々は，単純なハードウェア機構を用いて効果的に線形リスト中のデータをプリロード/プリフェッチする手法を新たに提案した [1]．本論文では，ポインタ操作が頻出するベンチマークプログラムを用いて，シミュレーションによる本方式の詳細な性能評価を行う．

以降，2節で本論文で扱うデータプリロード方式について述べる．続いて，3節でシミュレーションによる評価結果について報告する．4節で関連研究について述べ，最後に5節でまとめとする．

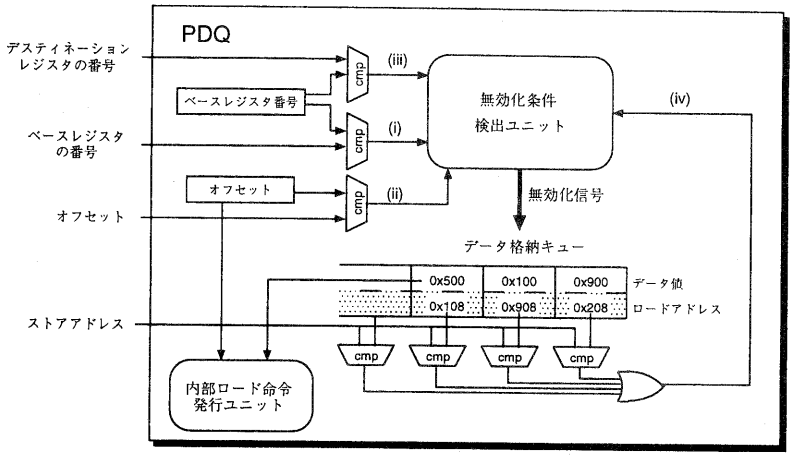

図 1: PDQ の構成

2 データプリロード方式

2.1 プリロード方式の概要

通常，線形リスト構造の各データ要素は，それぞれ複数のデータフィールドを保持しており，そのうち 1 つのデータフィールドに次のデータ要素の先頭アドレス（ポインタ）が格納されている．実際のプログラム中では，このポインタをたどることで連結された各データ要素に対して逐次アクセスする．このような処理において，本プリロード方式は，各データ要素のポインタを実際の参照に先行して次々とたどることによって，近い将来参照されるであろう各データ要素のポインタをプリロードする．

本プリロード方式を実装する場合，ハードウェアサポート機構として，図 1に示す PDQ(Prefech Data Queue) をプロセッサ内部に装備する．PDQ は，命令パイプライン中のデコードステージにおいて，実行時に以下の 2 つの条件を満たすロード命令を動的に検出する．

- ロードするデータの実効アドレスが 1 つの整数レジスタ[1]と定数のディスプレースメントによって指定されていること．

- 実効アドレスを指定するのに使われている整数レジスタが，デスティネーションレジスタとしても指定されていること．

[1]PDQ では，このレジスタを「ベースレジスタ」と呼ぶ

PDQ は，2 個のレジスタ，データ格納キュー，内部ロード命令[2]発行ユニット，そして無効化条件検出ユニットで構成される．PDQ が上記の条件を満たすロード命令を初めて検出した場合，そのロード命令のベースレジスタ番号とオフセットの値を PDQ 内部の 2 個のレジスタにそれぞれ記憶する．その後，ロードしたデータの値 (ポインタ) を PDQ 内部のデータ格納キューに一旦蓄える．続いて，内部ロード命令発行ユニットが，ロードした値とオフセットをソースデータに指定して内部ロード命令を自動的に生成し，命令テキスト中に現れるロード命令と同様に，プロセッサ内部のロードストアユニットに対して発行する．内部ロード命令発行ユニットは，データ格納キューのすべてのエントリが常に満たされるように繰り返しデータのプリロードを行う．このような処理の後，条件を満たすロード命令が再び検出されると，プリロード済みのデータをロード結果の値とし，ロードした値を使用する後続命令に対してメモリアクセスの遅延なしにデータを供給する．

以上のように，本プリロード方式は，ポインタ値を必要とする後続の命令をメモリアクセスの遅延なしで発行することが可能となるので，命令レベル並列処理の効率を高めることができる．また，各データ要素の先頭アドレスをプリロードすることで，そのデータ要素の他のデータフィールドの値も同時にキャッシュにロードされ，その後のデータフィールドのアクセスについても高速化が期待できる．

[2]PDQ 内でプリロード用に自動生成されるロード命令を，命令テキスト中に現れるロード命令と区別して，「内部ロード命令」と呼ぶ

2.2 プリロード済みデータの無効化条件

本プリロード方式では，線形リスト構造に変化がないものと仮定してポインタ値のプリロードを行う．従って，プリロード対象となっている線形リスト構造が変更された場合には，プリフェッチ済みデータの無効化を行う必要がある．本プリロード方式において，無効化検出条件は以下の4つであり，PDQにおいて，その内部に装備された無効化条件検出ユニットがこれらの条件を検出する．なお，図1中の無効化条件検出ユニットへの入力信号に，対応する以下の要件の番号を付している．

(i) ソースプログラムにおいてポインタ変数が異なる（従って異なるリストが処理されている）と推測できる場合． 当該ロード命令が2.1節で述べた条件を満たすが，ロードアドレスの計算に用いるソースレジスタの番号がPDQ内のベースレジスタ番号と異なる場合である．

(ii) 処理中のリストにおいて，異なるポインタをたどるものと推測できる場合． 当該ロード命令が2.1節で述べた条件を満たしており，ソースレジスタの番号もPDQ内のベースレジスタ番号と同一であるが，ロードアドレスの計算に用いるオフセットが異なる場合である．

(iii) ポインタ変数がポインタのたどり以外の方法で更新された場合，あるいは，ポインタ変数のレジスタ割り付けが変更されている場合． 当該ロード命令以外の命令のデスティネーションレジスタが，当該ロード命令でソースレジスタ（デスティネーションレジスタ）として使用しているレジスタと一致している場合である．

(iv) リスト構造が変更された場合 当該ロード命令に先行するストア命令の実行により，ポインタの値が格納されているメモリアドレスに書き込みが行われた場合である．

PDQは，(i)～(iii)の場合については，デコードステージにおいて検出を行い，PDQ内のキューを無効化する[3]．

(iv)の場合は，ストア命令の実効アドレスの計算が完了した後，その実効アドレスとPDQ内に保持しているプリロードデータのアドレスとの比較を行う．このアドレス比較処理の間にPDQ内のプリロードデータを使用して投機実行した後続の命令は無効化し，再実行する．

投機実行の失敗からの回復を行うために，リオーダバッファのエントリに，2つのフラグを追加する．一方は，ストアアドレスが，データ格納キュー内のロードアドレスと一致しているかどうかを表すフラグ(S-flag)であり，もう一方は，プリフェッチ済みデータをロード結果として使用したロード命令を表すフラグ(L-flag)である．リオーダバッファのエントリの先頭に，ストア命令が現れた場合，まずS-flagをチェックする．S-flagがT(True)の場合，このストア命令により，PDQが誤ってプリフェッチを行っている可能性があるので，その検出結果を蓄積する．その後，L-flagがTのロード命令がリオーダバッファの先頭の位置に到達したとき，それまでにS-flagがTのストア命令が現れていればリオーダバッファをフラッシュし，そのロード命令から再実行を行う．

[3](i), (ii)については，新たに当該ロード命令のソースレジスタの番号とオフセットをPDQ内の2つのレジスタに格納し，データ格納キューの先頭のエントリからデータのプリロードを再開する．

3 性能評価

3.1 評価プログラムおよび評価条件

本論文では，評価に用いるアプリケーションプログラムとしてOlden benchmark suite [2]を使用した．このベンチマークプログラムには全10種類のポインタ操作を多用するプログラムが含まれている．この中から，本論文の対象とする線形リスト構造を含むプログラムを選択し，評価に用いた．選択したアプリケーションプログラムを表1に示す．参考のため，表中には，全実行命令列中で検出対象ロード命令が何命令間隔で出現したかを測定し，その平均をとったものと，検出対象ロード命令が全実行ロード命令に占める割合の2種類のデータを示した．

3.2 評価結果

3.2.1 性能向上率

シミュレーションによる評価結果を図2に示す．図2のグラフは，本プリロード方式を実装しない場合を基準とした全実行サイクル数における性能向上率を示している．また，PDQ内のデータ格納キューにプリロードされたデータがロード値として使用された割合(PDQヒット率)を図3に示す．これらの図中，S4，S8はそれぞれ4命令，8命令同時発行の場合を示す．

本方式を用いることで，すべてのプログラムにおいて性能が向上しているが，データ格納キューの深さが2の場合の性能向上は大きくない．これは，2つのポインタを先行してたどる程度では，プリロー

表 1: アプリケーションプログラム

プログラム	入力パラメータ	データ構造	検出対象ロード命令の割合 [%]	検出対象ロード命令の間隔 [命令]
em3d	2000 nodes	線形リスト	6.29	46.80
health	3 levels, 1000 iters	4 分木，線形リスト	30.54	10.02
mst	256 nodes	ハッシュ，線形リスト	20.34	21.54
tsp	10000 cities	2 分木，線形リスト	15.46	31.27

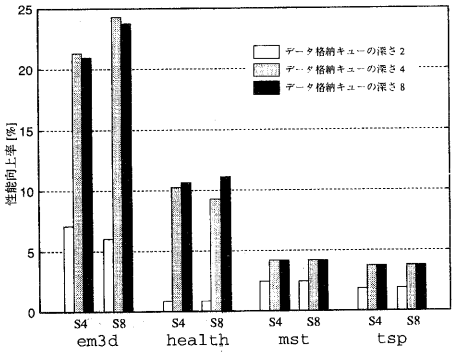

図 2: 性能向上率

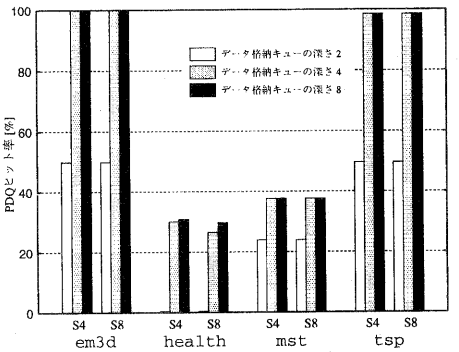

図 3: PDQ ヒット率

ド処理を実際のループ本体の実行とを完全にオーバラップできず，メモリアクセス遅延を隠蔽するのに十分なループレベルの並列性を抽出できていないためである．同時命令発行数については，4 命令発行の場合よりも，8 命令発行の場合の方が性能が向上する．これは，プリロードしたロード値を後続の命令に対して遅延なく供給することで命令並列度が向上することに加え，同時命令発行数を増加させることで，プログラムの全実行時間が短縮され，本方式によるメモリアクセス遅延を隠蔽する効果が，相対的に大きくなるためである．

すべてのプログラムにおいて 1 次キャッシュのヒット率は，0.5%～6.5%向上する．一方，2 次キャッシュについても，わずかにヒット率が向上するが，tsp のみ約 2%低下する．これはポインタのアドレスに対して先行してアクセスすることで，マッピングコンフリクトが発生したためと考えられる．また，データ格納キューの深さについては，深さ 8 の場合には深さ 4 の場合よりもわずかにヒット率が低下する．

以下，各プログラムごとに詳しく考察する．

em3d は，全プログラム中で最も性能向上が大きい．キューの深さが 4 以上の場合は，全検出対象ロード命令のうち 99%以上のデータが PDQ から供給されている．ループ本体の実行時間は比較的長いため，プリロード処理と実際のループ本体の実行とがオーバラップされ，ポインタ値をロードするためのメモリアクセス遅延がほぼ完全に隠蔽されている．

health は，em3d に比べて PDQ ヒット率が低い．health は，ループ本体の実行時間が短いので，プリロードすべきデータが間に合わず，ロード値として供給できないためである．データ格納キューの深さが 2 の場合はこの傾向が顕著に現れ，性能はほとんど向上しない．その一方で，health は他のプログラムと異なり，対象ロード命令の出現頻度が高く，また，数十回以上ループして線形リストをたどるものが多数出現する．これにより，全体として良好な性能を得ることができている．

mst は，PDQ ヒット率は高いが，対象ロード命令の出現頻度が低く全体の性能向上は低い．また，1 つのリストをたどるループ内で別のリストをたどり始めるので，PDQ 内のデータ格納キューの無効化が他のプログラムに比べ頻繁に発生し，これによる性能低下が起こる．

最後に，tsp についてであるが，PDQ ヒット率も高く，プリロード処理は効果的に機能している．しかし，他のプログラムとは異なり，2 次キャッシュのヒット率が低下する．性能向上の要因はポインタ

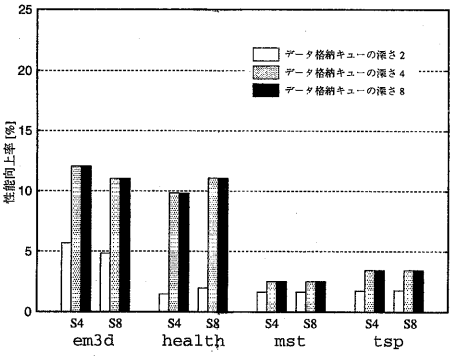

図 4: プリロードの効果 (キャッシュヒット率 100%)

値のプリロードによる効果のみで，2 次キャッシュへのプリフェッチによる効果が得られず，全体としての性能の向上は大きくない．

以上のように，本プリロード方式を実装することによる，命令並列度の向上とキャッシュのヒット率への影響が互いに関連し，全体の性能を決定していることがわかる．また，データ格納キューの深さについては，深さ 2 の場合では本プリロード方式の効果を得るには不十分であることが明らかとなった．

以上の結果を総合的に判断すると，データ格納キューの深さは，その深さに対するハードウェア量の投資を考慮すると，4 の場合が最も効果的であることが明らかとなった．

3.2.2 プリロードの効果

本プリロード方式の効果は，プロセッサの性能向上に対して 2 つの要素に分けて考えることができる．1 つは，ポインタ値を PDQ 内に蓄積し，後続の命令に対して遅延なくデータ値を供給することによる効果 (プリロード効果) であり，もう 1 つは，メモリの上位階層に各データ要素のポインタ値以外のデータフィールドが同時にキャッシュにロードされ，その後のデータフィールドのアクセスについて高速化すること (プリフェッチ効果) である．本節では，プロセッサ内にデータ格納キューを装備することによって可能となるプリロード効果について検討する．

プリフェッチ効果を除くために，データキャッシュが常にヒットするものとしてシミュレーションを行った．図 4にその結果を示す．このグラフは，本プリロード方式を実装しない場合を基準とした全実行サイクル数における性能向上率である．

em3d を除くプログラムにおいては，プリロード効果による性能向上が大きく，データキャッシュのシミュレーションを行う場合の結果に比べて，同等の性能向上を得ることができる．em3d については，図 2のグラフと比較すると，プリロードのみによる効果は薄れている．このプログラムでは，ループのイタレーション内で，ポインタ値をロードした直後に，他のデータフィールドの値をロードする．この処理を行うロード命令に対するプリフェッチによる効果が実際には大きいと考えられる．

以上から，本プリロード方式において，プリロード効果は大きく，PDQ 内にデータ格納キューを装備することの意義は大きいと結論できる．

3.2.3 PDQ 内のデータの無効化

2節で述べた PDQ 内のデータを無効化する 4 つの条件が，どのような割合で発生するかについて調査した．本節ではこの結果について述べる．

図 5に各プログラムごとの無効化条件の発生割合を示す．この図は，8 命令同時発行，データ格納キューの深さ 4 の場合のものである．本来，無効化の条件は 4 つしかないが，詳細な調査を行うために，ベースレジスタとオフセットの値がともに異なる場合 (これを (v) と記す) を，それぞれが単独で異なる場合とは別に集計した．

全体的には，条件 (iii) による無効化の割合が多い．これは，プログラム中においては，あるリストに対してポインタ値をたどった後，別のリストに対して処理を行う際に，メモリから当該リストの先頭アドレスをロードする場合に相当する．

また，次に発生頻度が高い条件 (i)，(ii)，(v) は，あるリストをたどっている途中で，レジスタ割り当ての異なる別のリストに対してたどりを開始した場合に発生する．本プリロード方式は，単一のリストをたどる場合にしかプリロードの効果は得られないために，この種の無効化が頻繁に発生する．

図 5からわかるように，今回選択した 4 種類のプログラムにおいては，条件 (iv) による無効化は発生しない．実際には，health や mst において，条件 (iv) で想定しているリスト構造に対する変化が生じる．これは，線形リスト構造に対して検索操作を行い，発見されたデータ要素の削除を行う部分である．しかし，この処理では，検索操作を行うループにおいて，本方式によりプリロードした値を使用し，実際にポインタ値が格納されているアドレスに対して書き込みが発生するのは，ループを抜けた後，当該データ要素の削除が行われるときである．従って，誤ったプリロードデータを用いて投機実行に失敗することはあり得ない．

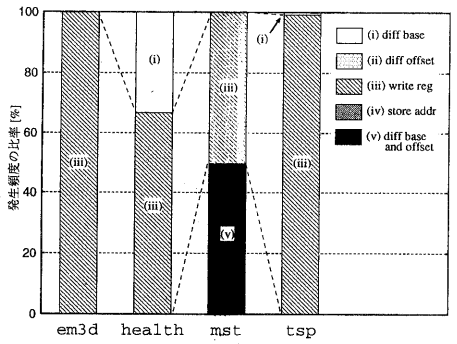

図 5: 無効化条件の発生割合 (8 命令同時発行，データ格納キューの深さ 4 の場合)

以上のように，本実験結果により，本プリロード方式の性質として，複数のリストをたどる場合に無効化が多く発生し，性能向上を制限していることが明らかとなった．また，性能に悪影響を与えると懸念された投機実行失敗の影響は，実際のプログラムでは存在しないことが確認できた．

4 関連研究

線形リスト構造に対してデータプリフェッチを行う方式として，Dependence-based prefetching(DBP と略記) [3] が提案されている．この方式は，本プリロード方式と同様に，各データ要素に含まれる，次の要素へのポインタ値を読み込むロード命令を実行時に検出する．プリフェッチしたデータを一時的に格納する Prefetch Buffer(PB) をプロセッサ外部に装備し，キャッシュメモリに対してプロセッサと並列にアクセスし，データをプリフェッチする．ポインタ値をロードする命令は，PB からデータを読み出す際，キャッシュメモリと同じアクセス遅延を伴う．これに対し，本プリロード方式は，ポインタ値をプロセッサ内部のキューにプリロードし，命令パイプラインの実行ステージに遅れなくデータの供給を行うことが可能である．今回の実験で，このプリロード効果が大きいことを確認した．

また，ハードウェアサポート機構の規模については，DBP では，PB やキャッシュメモリに多数のアクセスポートを設ける必要があり，またそれらの制御も複雑で大規模なものとなる．これに対し，本プリロード方式は，単純なハードウェアサポート機構しか必要しないにもかかわらず，データ格納キューの深さ 4 の場合，1KB もの大きさの PB を装備する DBP と比べても遜色のない性能向上を得ることができている．

5 おわりに

本論文では，ポインタ操作が頻出するアプリケーションプログラムを用いて本プリロード方式の性質について詳細な評価を行った．

評価の結果，本プリロード方式を実装することで 3.7%～24.3% の性能向上を得た．今回の実験では，プロセッサ内部にデータ格納キューを設けデータをプリロードすることで 2.5%～12.0%の性能向上を得ることが可能であり，その効果が大きいことを確認した．

本プリロード方式では，誤ってプリロードしたポインタ値を用いて投機実行した命令を無効化し，再実行しなければならない．しかし，今回の実験で，実際のアプリケーションでは再実行による影響は極めて小さく，性能低下の要因にはなり得ないことを確認した．

今後は，本方式の適用範囲拡大のための検討を行う予定である．

謝辞

本研究の一部は，文部省科学研究費補助金 (10480062, 11480069, 11878052, 12780218) の補助による．

参考文献

[1] 山村 周史，平田 博章，新實 治男，柴山 潔，“線形リストを対象としたデータプリフェッチ機構”，並列処理シンポジウム JSPP2000，pp. 115–122, 2000.

[2] A. Rogers, M.Carlisle, J.Reppy, and L.Hendren: “Supporting dynamic data structures on distributed memory machines.”, *ACM Transactions on Programming Languages and Systems*, Mar. 1995.

[3] A. Roth, A. Moshovos, and G. S. Sohi: “Dependence Based Prefetching for Linked Data Structures”, *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems*, 1998.