# 擬似ベクトルプロセッサにおける高速リストベクトル処理

廣野 哲[†1] 中村 宏[†2]
朴 泰祐[†3] 中澤 喜三郎[†4]

大規模科学技術計算においては，データ参照に時間的局所性が少ないためにキャッシュが有効に働かない．このような計算においても高い実効性能を達成する擬似ベクトルプロセッサPVP-SWを我々は提案している．また，データがランダムに参照され，データ参照に空間的局所性も少ないリストベクトル処理においてもPVP-SWは有効であることが過去に報告されている．しかし，過去の報告では，リストベクトルの内容に重複がないことが保証された場合のリストベクトル処理についてのみ論じている．本論文では，このような保証がなく，従来のベクトル型スーパーコンピュータではベクトル化できない一般のリストベクトル処理においてもPVP-SWが効率良く処理を行えることを示す．計算機シミュレーションによる性能評価結果より，PVP-SWが高い実効性能を達成することが確認できた．

# Fast List Vector Computation on Pseudo Vector Processor

AKIRA HIRONO,[†1] HIROSHI NAKAMURA,[†2] TAISUKE BOKU[†3]
and KISABURO NAKAZAWA[†4]

In large scientific/engineering applications, data caches do not work effectively because of little temporal locality. We have proposed "Pseudo Vector Processor based on Slide-Windowed Registers (PVP-SW)" for these applications. This processor realizes high performance even in list vector processing which has little spatial locality due to random data accesses. However, previous reports assumed that none of the list vector data is the same. In this paper, we focus on more general list vector computation without this assumption. Such list vector computation can not be vectorized in ordinary vector supercomputers. We show that PVP-SW is also effective even in such list vector processing. Performance evaluation reveals that PVP-SW achieves high performance even in general list vector processing.

## 1. はじめに

近年，集積回路技術の進歩によるクロック周波数の向上，スーパースカラ等の命令レベルの並列性を抽出して高速に処理を行う処理方式の実用化により，スカラプロセッサの処理性能は飛躍的に向上している．しかし，プロセッサに対するデータ供給系に関しては，メモリ素子の集積度の向上により主記憶の大容量化が進んではいるものの，そのアクセスタイムはプロセッサのマシンクロックの向上と比べてさほど改善されてはいない．そのため主記憶アクセスレーテンシがプロセッサの実効性能に与える影響は相対的に増大する傾向にある．

この問題に対処するため，スカラプロセッサでは通常キャシュメモリを用いている．しかしながら，大規模な科学技術計算においては，データ領域は膨大であるものが多く，データ参照の時間的局所性も少ないため，キャッシュは有効には働かない．このためキャッシュミスが頻繁に起こり，主記憶アクセスペナルティのためスカラプロセッサの性能は著しく低下する．

一方，ベクトル型スーパーコンピュータにおいては主記憶アクセスはパイプライン化されており，そのスループットは高い．また，多数のベクトルレジスタを持つことによりベクトルロード処理のベクトル長を長くでき，チェイニング機構によりこれらの処理と演算処理を並列に行えることから，レジスタへのプリロード機能を実現できる．このため，主記憶アクセスレー

†1 株式会社日立製作所汎用コンピュータ事業部
General Purpose Computer Division, Hitachi Ltd.
†2 東京大学先端科学技術研究センター
Research Center for Advanced Science and Technology, University of Tokyo
†3 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba
†4 電気通信大学情報工学科
Department of Computer Science, University of Electro-Communications

テンシが性能に与える影響は低減される．

したがって，命令レベルの並列実行をねらったスーパースカラプロセッサにおいても主記憶スループットを高くし，主記憶アクセスレーテンシを隠蔽することができれば，データ供給命令と演算命令の並列実行により，演算パイプラインを切れ目なく稼働させることが可能であると考えられる．このようにして，ベクトル命令の処理内容を複数のスカラ命令によって擬似的に処理する手法を我々は擬似ベクトル処理[1]と呼んでいる．また，この擬似ベクトル処理を行うプロセッサとして擬似ベクトルプロセッサ PVP-SW (Pseudo Vector Processor based on Slide-Windowed Registers) を提案している．これは現在開発中の超並列計算機 CP-PACS[2]のノードプロセッサに採用されている．

大規模な科学技術計算における PVP-SW の有効性は文献3) ですでに示されている．また，データの参照がランダムアクセスになり，データ参照の空間的局所性も少ないリストベクトル処理においても，PVP-SW は良い性能を示すことが文献4) で報告されている．しかし，文献4) での評価は，リストベクトルの内容に重複がなく，リストによって間接参照されるデータ間に依存関係がないことが保証された場合のリストベクトル処理のみを対象としていた．

本論文では，データ間の依存関係がないという保証がなく，従来のベクトル型スーパーコンピュータではベクトル化できない一般のリストベクトル処理を対象とする．そして，このようなリストベクトル処理を PVP-SW 上で効果的に実行する手法を示す．さらに計算機シミュレーションにより，PVP-SW が高い性能を達成することを示す．

## 2. 擬似ベクトルプロセッサ PVP-SW

PVP-SW[3]は，従来のスーパースカラ方式のプロセッサに以下の機能拡張を施すことによって実現される．

(1) 主記憶アクセスのパイプライン化
(2) キャッシュメモリへのプリフェッチ機能の追加
(3) 浮動小数点レジスタへのプリロード/ポストストア機能の追加
(4) 浮動小数点レジスタ数の増加にともなうレジスタのスライドウィンドウ化

主記憶のスループット向上は大規模科学技術計算を高速に行ううえで必要不可欠であるため，(1) の拡張を行う．これは主記憶をマルチバンク化し，アドレス付けをインタリービングすることによって擬似的に実現される．

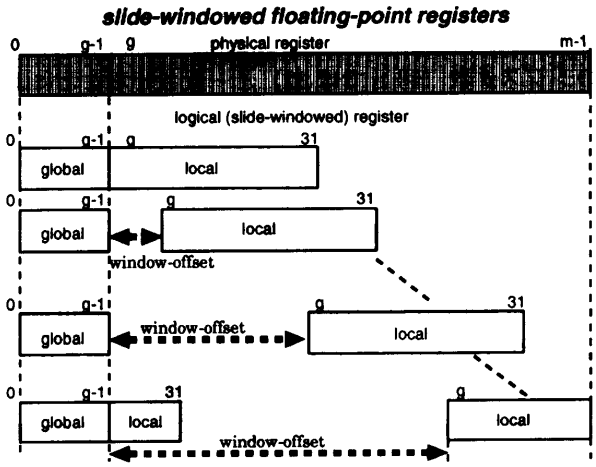


図1 浮動小数点レジスタのスライドウィンドウ構成
Fig. 1 Structure of slide-windowed registers.

(2) のプリフェッチ機構は，データをキャッシュへあらかじめ転送するプリフェッチ命令を用意するもので，近年のスカラ方式のマイクロプロセッサでは活用されつつあるものである．これに対し，(3)，(4) は PVP-SW に固有の機能拡張である．

(3) のプリロード/ポストストア機能[3]は，キャッシュミス時にはキャッシュメモリを介することなく，直接主記憶–浮動小数点レジスタ間でパイプライン的にデータ転送を行うものである (2.2.1 項参照)．プリロード機能を利用して長い主記憶アクセスレーテンシを隠蔽するには，プリロードされるデータの格納先である浮動小数点レジスタが多数必要となる．しかし，既存の命令セットでは命令フォーマット中のレジスタ指定フィールドの関係上，既存の命令セットアーキテクチャで決定されているレジスタ数以上のレジスタを参照することはできない．そこで PVP-SW では，(4) の浮動小数点レジスタのスライドウィンドウ化[3]を行っている．

### 2.1 浮動小数点レジスタのスライドウィンドウ構成

浮動小数点レジスタのスライドウィンドウ構成を図1に示す．物理レジスタ空間は複数の論理レジスタウィンドウに分割される．ウィンドウの位置は window-offset によって指定される．各ウィンドウは global 部と local 部に分かれ，global 部のレジスタはすべてのウィンドウで共通である．

このウィンドウ構成では，ある時点において有効な (アクティブな) ウィンドウは1つである．また，このアクティブ・ウィンドウはソフトウェアにより window-offset を指定することで物理レジスタ空間上を自由に移動 (スライド) することが可能である．拡張前のアーキテクチャにおける命令でのレジスタ指定では，アク

ティブ・ウィンドウ内の論理レジスタ番号を使用する．ウィンドウ内のレジスタ数を拡張前のアーキテクチャのレジスタ数（図1では32）と一致させることにより，拡張前の命令セットアーキテクチャと上位互換性を保ちながら，物理的により多くの浮動小数点レジスタを利用することが可能になる．

### 2.2 主記憶アクセスレーテンシの隠蔽手法

PVP-SWでは，プリロード/ポストストア機能のほかにデータキャッシュへのプリフェッチ機能を利用して主記憶アクセスレーテンシを隠蔽することもできる[4]．

#### 2.2.1 プリロード/ポストストア機能

プリロード/ポストストア機能はプリロード/ポストストア命令によって実行される．プリロード命令は，データをメモリから任意に指定されるウィンドウ内のレジスタにロードする．ポストストア命令は，データを任意に指定されるウィンドウ内のレジスタからメモリへストアする．

通常のロード/ストア命令と同様に，これらの命令がキャッシュヒットする場合，データはキャッシュ-レジスタ間で転送される．しかし，通常のロード/ストア命令とは異なり，キャッシュミスの場合には，データはキャッシュを介さずに主記憶とレジスタ間で直接転送される．また，キャッシュミスを起こしてもプロセッサの内部処理はストールせず，引続き後続命令は実行される．このため，プリロード/ポストストア命令による主記憶アクセスに長い時間を要したとしても，これらの命令と依存関係がない後続命令は実行可能である．したがって，ロードに関しては前もってプリロード命令を先行発行しておき，そのデータがレジスタに転送されるまでの間は依存関係のない他の命令を処理すれば，該当するデータの主記憶アクセスレーテンシを隠蔽することが可能となる．ストアに関しては，ストア・バッファを利用した置いてきぼり制御により，データのストア完了を待たずして後続の命令を実行することができる．

ベクトル型スーパーコンピュータにおけるベクトルロード命令の機能は，PVP-SWでは各ベクトル要素に対するプリロード命令を要素数分発行することで擬似的に実現される．しかも，これらのプリロード命令はスーパースカラ構成の場合には他の演算命令等と同時に実行させることができるので，ベクトルプロセッサのベクトルロード命令と等価な動作をすることができる．ベクトルロード命令とプリロード命令の違いは，前者はスカラレジスタ，キャッシュを用いないのに対し，後者はそれ自身はスカラ処理であり，他のスカラ命令とレジスタ，キャッシュを共有する点である．

#### 2.2.2 プリフェッチ機能

プリロード命令は浮動小数点レジスタへのロードのみを行うため，プリロード機能では浮動小数点データのアクセスレーテンシのみしか隠蔽できない．これに対し，キャッシュへのプリフェッチ機能では，浮動小数点データのみならず，整数データの主記憶アクセスレーテンシの隠蔽も可能である．

PVP-SWではソフトウェア制御によるプリフェッチ機能[5),6)]を採用する．これは，プリフェッチ命令を用いて，将来アクセスされるデータをあらかじめキャッシュに転送しておき，後に通常のロード/ストア命令を用いてキャッシュ-レジスタ間でデータ転送を行う手法である．プリフェッチ命令がキャッシュミスを起こしても，引続き後続命令は実行可能である．また，プリロード/ポストストア命令が浮動小数点データサイズ単位で処理されるのに対し，プリフェッチ命令では，対応するキャッシュライン分のデータが主記憶からキャッシュに転送される．

## 3. PVP-SWにおけるリストベクトル処理手法

ここでは図2に示すリストベクトル処理例を用いて，PVP-SWにおけるリストベクトル処理手法を説明する．

図2の処理では，1つの配列要素 $A(L(I))$ を参照するのに，リストベクトル $L(I)$ の参照と，$L(I)$ の値を用いた $A$ の間接参照の2つのメモリアクセスが必要となる．この2つのアクセスは逐次的に行う必要があるため，主記憶アクセスペナルティが性能に与える影響は大きくなる．$\{\forall I, J; I \neq J \Rightarrow L(I) \neq L(J)\}$ が成立する場合，すなわちリストベクトルの内容に重複はないことが保証されている場合は，PVP-SW上で効率良く実行する手法が文献4)ですでに示されている．しかし，$\{\exists I, J; I \neq J \wedge L(I) = L(J)\}$ が成立する場合，すなわち，リストベクトル $L$ に重複がないという保証がない場合には，$B$ に関して依存関係が生じる可能性がある．たとえば，図2の処理において $L(10) = L(11)$ の場合，$B(L(11))$ の値のロードは $B(L(10))$ の値のストアの後でないと正しい値が得られない．このような一般のリストベクトル処理においてはベクトルプロセッサでもベクトル化できない[7)]．

```
DO I=1,N
  B(L(I)) = A(L(I)) + B(L(I))
ENDDO
```

図2 リストベクトル処理例
Fig. 2 A example of list vector computation.

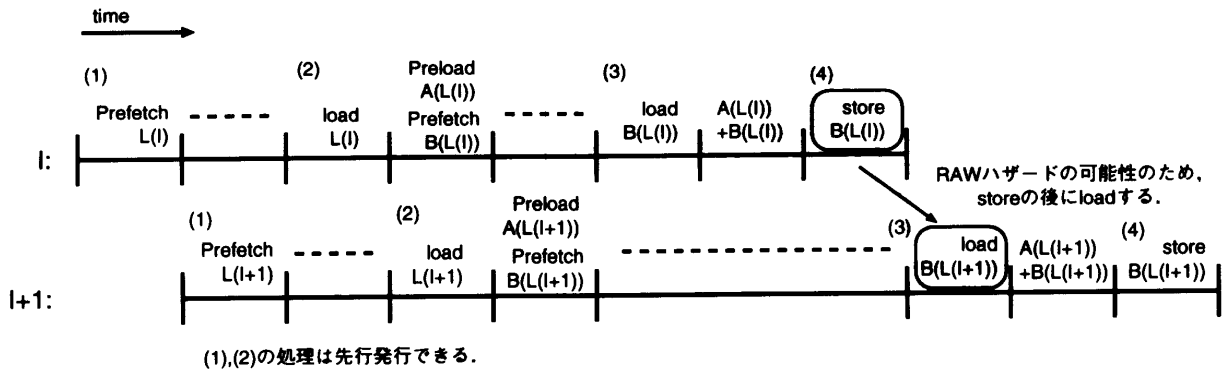


図3 $B(L(I)) = A(L(I)) + B(L(I))$ の処理タイミング
Fig. 3 Timing of calculation $B(L(I)) = A(L(I)) + B(L(I))$.

PVP-SW では，図2の各イタレーションの処理を次のような4ステップに分割して行う．

(1) 整数データであるリストベクトル $L(I)$ の値をキャッシュにプリフェッチする．

(2) $L(I)$ の値を通常のロード命令を用いてキャッシュから汎用レジスタにロードする．その値を用いて $A(L(I))$ を浮動小数点レジスタにプリロードする一方，$B(L(I))$ はキャッシュにプリフェッチする．

(3) $B(L(I))$ を通常のロード命令を使用してキャッシュから浮動小数点レジスタへロードする．すでにレジスタにある $A(L(I))$ と，今ロードした $B(L(I))$ を用いて演算を行う．

(4) 汎用レジスタに残っている $L(I)$ の値を用いて，演算結果 $B(L(I))$ を通常のストア命令を用いてストアする．

この4つのステップをイタレーション間では“前のイタレーションの step (4) が終了した後に現在のイタレーションの step (3) を実行する”という制約のもとに，**図3** に示すようにソフトウェア・パイプライン的に処理する．

この手法の本質は，リストベクトル $L$ と依存関係の生じる可能性のある配列 $B$ に対してはプリフェッチ，依存関係が生じる可能性のない配列 $A$ に対してはプリロードする点にある．一般にリストベクトル自身は連続参照されることが多く（そうでない場合も連続参照になるようにリストベクトルを変更することは容易である），プリフェッチ命令によりキャッシュライン分の複数個のデータを一括して転送することは効果的である．それに対し，リストベクトルを用いて間接参照される配列はランダムにアクセスされるため，プリフェッチすると同一キャッシュライン中の不要なデータも転送され，主記憶のトラフィックを増大させてしまう．したがって，間接参照される配列のアクセスに対しては，必要なデータのみを直接レジスタへ転送するプリロード命令を用いた方が効果的である．しかし，依存関係の生じる可能性のある配列については，直前のイタレーションの結果を直後のイタレーションで使用することがあるため，多少の主記憶スループットを浪費しても，アクセスレーテンシの短いキャッシュにそのデータを置く方が効果的である．

## 4. 評価環境

### 4.1 評価モデル

#### 4.1.1 プロセッサモデル

リストベクトル処理における PVP-SW の有効性を確認するために，以下の3つのプロセッサモデルに対しシミュレーションによる評価を行った．

**<Original>** PVP-SW への拡張を行う前のスーパースカラプロセッサモデルである．命令セットアーキテクチャの例として HP 社の PA-RISC 1.1 Architecture[8] を採用する．浮動小数点レジスタ数は32個である．主記憶はパイプライン化されておらず，キャッシュミス時には後続の命令は主記憶アクセスが完了するまでストールする．主記憶アクセスレーテンシ隠蔽のためのプリフェッチ，プリロード/ポストストア機能は有していない．

**<Prefetch>** 主記憶アクセスレーテンシを隠蔽する方法として，プリフェッチ命令によってデータをキャッシュにプリフェッチする方法のみを採用するプロセッサモデルである．<Original> にプリフェッチ機能を追加したモデルである．主記憶はパイプライン化されており，プリフェッチ命令による主記憶アクセスはパイプライン的に処理されるが，通常のロード/ストア命令によるアクセスでは，キャッシュミス時に後続命令はストールする．

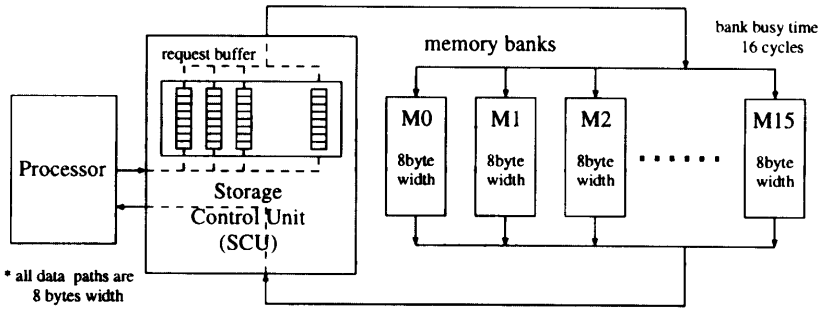


図4 インタリーブ・メモリの構成
Fig. 4 Structure of interleaved memory.

**<PVP-SW>** 我々が提案している擬似ベクトルプロセッサである．<Prefetch> に浮動小数点レジスタのスライドウィンドウ化とプリロード/ポストストア機能を追加したモデルである．浮動小数点レジスタ数は 128 個に増加してある．プリフェッチ命令，プリロード/ポストストア命令によるアクセスはパイプライン化されている．

また，全モデルに対して以下の共通な仮定をおいた．

- 同時命令発行数は 2 命令とする．整数演算系の命令は同時に 2 命令発行可能だが，ロード/ストア系（プリフェッチ，プリロード/ポストストア命令を含む），浮動小数点演算，分岐命令の 3 つのグループの命令に関しては，同一グループ内の 2 命令を同時発行することはできないものとする．
- 整数演算レーテンシは 1 MC (Machine Cycle)，浮動小数点演算レーテンシは 2 MC とする．
- 命令キャッシュは必ずヒットするものとする．
- データキャッシュは direct map 方式の 1 次キャッシュのみを想定する．キャッシュサイズは 256 Kbyte，ラインサイズは 32 byte とする．キャッシュヒット時のアクセスレーテンシは 1 MC とする．

#### 4.1.2 主記憶モデル

<Original> においては，主記憶アクセスはパイプライン化されていない．キャッシュミス時の主記憶アクセスレーテンシは 40 MC とする．<Prefetch>，<PVP-SW> における主記憶には，マルチバンク構成のインタリーブ・メモリを仮定する．

**図 4** に仮定するインタリーブ・メモリのモデル図を示す．すべてのデータ幅は 8 byte であり，各メモリバンクのデータ幅も 8 byte である．また，各バンクのアクセスに要する時間（bank busy time）は 16 MC とする．バンク数が 16 バンクなので，バンクコンフリクトが起こらない場合に最高 8 byte/MC の主記憶スループットを実現する．メモリ・リクエストは 8 byte 単位で行われ，そのアクセスには 40 MC 要するとする．しかしバンクコンフリクトが起こると，アクセスレーテンシに最高 16 MC のペナルティがかかる．SCU (Storage Control Unit) は，各バンクごとに十分なサイズのリクエスト・バッファを持っており，これを用いて各バンクに出されるメモリ・リクエストを管理している．各バンクのアドレス付けは low-ordered interleaving 法を用いており，バンクの選択はアドレス情報の下位 4 bit を用いてなされる．

```
L1:  B(L(I)) = A(L(I)) + B(L(I))
L2:  B(L(I+2)) = A(L(I+1)) + B(L(I))
L3:  B(L(I+2)) = A(L(I+1)) * R + B(L(I))
L4:  B(L(I+3)) = A(L(I+2)) * C(L(I+1)) + B(L(I))
L5:  C(L(I+7)) = ((A(L(I+6)) + E(L(I+5))) +
                  (B(L(I+4)) + F(L(I+3))) -
                  (C(L(I+2)) + D(L(I+1)))) * G(L(I))
LFK 14: RH(IR(K)   ) = RH(IR(K)   ) + fw - RX(K)
        RH(IR(K)+1) = RH(IR(K) + 1) + RX(K)
```

図5 評価ベンチマーク
Fig. 5 Benchmark programs.

#### 4.1.3 プリフェッチ機構の仮定

<Prefetch>，<PVP-SW> においてはプリフェッチされたデータを主記憶からキャッシュに格納しつつ，CPU からのキャッシュの参照を許すように，データキャッシュのスループットは十分であるものとする．また，プリフェッチ・リクエストの状態を管理するためのリクエスト・バッファの数は十分にあり，発行されたプリフェッチ命令はすべてパイプライン的に処理されるものとする．

### 4.2 評価ベンチマーク

**図 5** に評価に用いたリストベクトル処理を示す．L1 ～L5 は文献 9) から，LFK 14 は Livermore Fortran Kernels より引用した．

配列 $A$～$G$，$RH$，$RX$ は倍精度浮動小数点データ，リストベクトル $L$，$IR$ は整数データである．配列 $A$～$G$，$RH$，$RX$ は各々キャッシュの容量を十分越えるサイズ（各 64 K 要素，512 Kbyte）とし，それに合わ

せてリストベクトルも拡張した．リストベクトルのサイズは32K要素で128Kbyteであるが，値は1～64Kの範囲になるようにした．このリストベクトルはLFK 14で用いられているリストベクトル作成アルゴリズムをもとに作成しており，$\{\exists I, J; I \neq J \wedge L(I) = L(J)\}$ の条件を満たしている．リストベクトル $L(I)$ を用いて $A(L(I))$ をアクセスしてみたところ，配列 $A$ の64K個要素のうち約30%が参照され，1要素への最高参照回数は9回であった．

各カーネルのターゲットプログラムはそれぞれのプロセッサモデルにおいて最適になるようにハンドコンパイルによって作成した．プリフェッチ命令を用いる <Prefetch>，<PVP-SW> においては，ループ・アンローリング手法を用いてできるだけプリフェッチ命令の数が少なくなるようにした．また，バンクコンフリクトの影響を最小限に抑えるようなメモリ・アロケーションを行った．

## 5. 評価結果と考察

### 5.1 評価結果

各ベンチマークプログラムを各々のプロセッサモデルで実行させたときの実行時間を計算機シミュレーションによって測定した．図6に各プロセッサモデルの性能を示す．性能指標として，FLOPC (FLoating Operations Per clock Cycle) を用いた．

<Original> は主記憶アクセスレーテンシ隠蔽の機構を何ら持たないので，他のプロセッサモデルと比べて著しく性能が低い．<Prefetch> は <Original> よりも良い性能を示すが，<PVP-SW> よりは性能が低い．<PVP-SW> はどのプロセッサモデルよりも良い性能を示し，<PVP-SW> は <Prefetch> の約1.1～2.3倍の性能を示した．

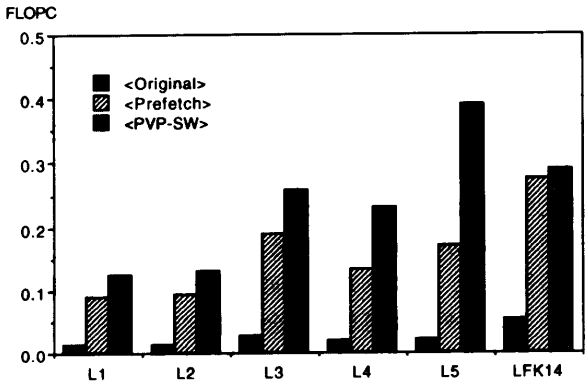


図6 各プロセッサモデルの性能
Fig. 6 Performance of each processor model.

### 5.2 <Prefetch> と <PVP-SW> の比較

<Prefetch> と <PVP-SW> の性能差の原因を調べるため，主記憶，キャッシュモデルを変更して性能評価を行った．その結果を図7に示す．

#### 5.2.1 プリフェッチ命令による影響

図7中の <all_hit> はデータキャッシュが必ずヒットすると仮定した場合の理想的な性能を表している．<Prefetch/all_hit> と <PVP-SW/all_hit> の性能差は，依存関係が生じない，ランダムアクセスされる配列の数による．このような配列は，L3においては配列 $A$ のみの1種類であるが，L4では $A$, $C$ の2種類，L5では $C$ 以外の6種類である．依存関係が生じない，ランダムアクセスとなる配列のアクセスには，プリフェッチ命令よりもプリロード命令が有効に働く．このような配列の数が多いと，<Prefetch> と比較した場合の <PVP-SW> の有効性が顕著になる．L3においては <PVP-SW/all_hit> は <Prefetch/all_hit> より13.5%性能が良く，L4では27.1%，L5では88.4%性能が良い．

<Prefetch> では，参照するデータすべてをプリフェッチする．LFK 14の配列 $RX$ のように，プリフェッチされるデータが連続アクセスになる場合には1つのプリフェッチ命令で計算に必要なデータを複数個キャッシュに転送することができる．そのためループ・アンローリング手法を用いれば，<Prefetch> においても，実行されるプリフェッチ命令数を減らすことができる．しかし，配列 $RH$ のようにリストベクトルを用いた間接参照では，そのアクセスは一般にはランダムになるため，プリフェッチ命令数を減らすことはできない．<PVP-SW> では，連続アクセスになるリストベクトルと依存関係が生じる可能性があるデータのみをプリフェッチし，依存関係が生じないデータに関しては間接参照のためランダムアクセスになるデータでもプリロード命令でアクセスするので，プリフェッチ命令数は最小限にできる．プリフェッチ命令そのものの実行に時間がかかるため，依存関係が生じないランダムアクセスとなる配列データが存在するL1～L5においては，<Prefetch/all_hit> は <PVP-SW/all_hit> より性能が低くなるのである．

#### 5.2.2 バンクコンフリクト，ラインコンフリクトによる影響

<PVP-SW> は <PVP-SW/all_hit> の80～90%の性能を達成しているが，<Prefetch> においては <Prefetch/all_hit> の62～86%の性能しか出ていない．性能低下の原因としては，

- バンクコンフリクトによる主記憶スループットの

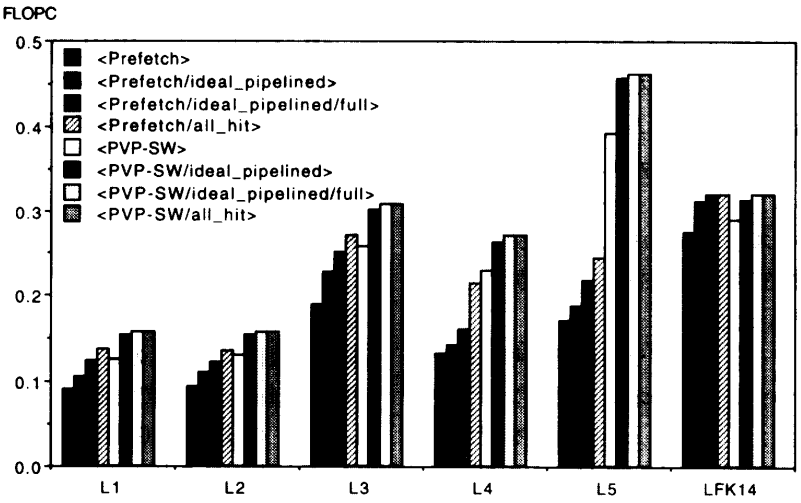


図7 <Prefetch> と <PVP-SW> の性能
Fig. 7 Performance of <Prefetch> and <PVP-SW>.

低下と主記憶アクセスレーテンシの増加

- キャッシュのラインコンフリクトによる必要データの追い出し

が考えられる．これらの影響を調べるため，主記憶をバンクコンフリクトを生じない理想的なパイプライン・メモリとし，バンクコンフリクトの影響を排除した評価結果を図7に <ideal_pipelined> として示す．さらにラインコンフリクトの影響を調べるため，キャッシュを full associative 方式とした場合の評価結果を <ideal_pipelined/full> として示す．

図7より分かるように，<Prefetch/ideal_pipelined> は <Prefetch/all_hit> の性能に及ばない．バンクコンフリクトとラインコンフリクトの影響を排除した <Prefetch/ideal_pipelined/full> でも，理想性能である <Prefetch/all_hit> に及ばない．ほとんどのデータ参照がランダムアクセスになる L1～L5 においては，<PVP-SW> にも劣ってしまう．ランダムアクセスになるデータのプリフェッチでは，1キャッシュライン分のデータ（ここでは倍精度浮動小数点データ 4 個）がブロック転送されても，その全部が計算に用いられるわけではない．そのため計算に不必要な無駄なメモリ・トラフィックが多発し，主記憶スループットが不足して性能が低下するのである．また，<Prefetch/ideal_pipelined> と <Prefetch/ideal_pipelined/full> の性能差から，プリフェッチ命令を多数用いる <Prefetch> においては，direct map 方式のデータキャッシュでは必要なデータが他のデータのプリフェッチに際してラインコンフリクトによりキャッシュから追い出される cache pollution の問題が無視できないことが分かる．

一方，<PVP-SW> では，理想的なパイプライン・メモリを使用した <PVP-SW/ideal_pipelined> が <PVP-SW/all_hit> に近い性能を示す．これより，<PVP-SW> に必要な主記憶スループットは 8 byte/MC で十分であることが確認できる．しかし，<PVP-SW> の性能は <PVP-SW/ideal_pipelined> の 81～93% しか出ていない．これは，ランダムアクセスのために起こるバンクコンフリクトの影響で，実効的な主記憶スループットが低下していることを表している．また，<PVP-SW/ideal_pipelined> と <PVP-SW/ideal_pipelined/full> の性能にあまり差がないことから，<PVP-SW> では cache pollution 問題はさほど深刻ではないことが分かる．

### 5.3 <PVP-SW> とベクトル型スーパーコンピュータの比較

文献 9) では，図5で示した L1～L5 のリストベクトル処理をスーパーコンピュータで実行したときの実効性能が報告されている．

**表1** に文献 9) から抜粋したベクトルマシンでのリストベクトル処理における性能を示す．<PVP-SW> の性能は，本論文データを文献 2) で報告されている予定動作周波数 150 MHz で換算している．また，本論文で行ったシミュレーションのパラメータは，実装予定の仕様から選択している．各スーパーコンピュータの性能は，ベクトル長 5000，擬似乱数で生成したリストデータを使用した場合の性能である．

表1より分かるように，比較対象のスーパーコンピュータが一世代程度前のものとはいえ，<PVP-SW> は他のマシンよりも良い性能を示している．<PVP-SW> では，プリロード/ポストストア命令で使用す

表1 リストベクトル処理における性能 (MFLOPS)
Table 1 Performance on list vector computation (MFLOPS).

| | VP2600/10 | S-820/20 | SX-2N | <PVP-SW> |
|---|---|---|---|---|
| L1 | 5.2377 | 4.0890 | 1.5964 | 18.751 |
| L2 | 4.2883 | 2.4576 | 1.5726 | 19.474 |
| L3 | 7.9572 | 5.1782 | 3.0739 | 38.600 |
| L4 | 4.3976 | 4.8584 | 1.9724 | 34.446 |
| L5 | 3.6491 | 9.4881 | 2.1761 | 58.615 |

るレジスタも，キャッシュを介する通常のロード/ストア命令で使用するレジスタも，演算で使用するレジスタも，すべて同じ浮動小数点レジスタを使用する．このため，ベクトルロード処理に相当するプリロード命令を用いたロード処理とキャッシュを用いるスカラ処理を効率良く混在させて相補的に用いることができるという利点がある．一方，ベクトル型スーパーコンピュータでは，ベクトル処理で用いられるレジスタとスカラ処理で用いられるレジスタ，キャッシュは分離されている．これが通常はベクトル化できないリストベクトル処理でベクトル型スーパーコンピュータと比較して <PVP-SW> が良い性能を達成する理由としてあげられる．

## 6. おわりに

本論文では，データ参照に依存関係が存在する可能性がある場合のリストベクトル処理に着目し，PVP-SW 上での処理性能をシミュレーションによって測定した．

PVP-SW では主記憶アクセスレーテンシ隠蔽の方法として，キャッシュへのソフトウェア・プリフェッチ機能，プリロード/ポストストア機能の 2 種類の方法を利用できる．ソフトウェア・プリフェッチでは，

- プリフェッチ命令挿入による実行時間の増加
- cache pollution
- ランダムアクセス時の不要なデータのプリフェッチによるメモリ・トラフィックの増大

といった問題が生じる．PVP-SW のプリロード機能では計算に必要なデータのみをキャッシュを介さずに直接レジスタに転送できる．また，使用するプリフェッチ命令は通常は連続アクセスとなるリストデータ，および依存関係が存在する可能性があるデータに限られるので，プリフェッチによる諸問題を最小限に抑えることができる．

評価結果より，PVP-SW ではプリロード/ポストストア機能を利用したベクトル処理と，キャッシュを利用したスカラ処理の併用により，従来のベクトルプロセッサではベクトル化できないリストベクトル処理を効率良く扱うことができることが確認できた．

**謝辞** 本研究を進めるにあたり貴重なご意見をいただいた筑波大学西川博昭助教授，ならびにアーキテクチャ研究室諸氏に深く感謝します．なお，本研究は一部文部省科学研究費（創成的基礎研究 07NP0401，および奨励研究（A）07780222）による．

## 参考文献

1) Nakazawa, K., Imori, H., Nakamura, H. and Kawabe, S.: Pseude Vector Processor Based on Register-Windowed Superscalar Pipeline, *Proc. Supercomputing '92*, pp.642–651 (1992).
2) 中澤喜三郎，中村 宏，朴 泰祐：超並列計算機 CP-PACS のアーキテクチャ，情報処理，Vol.37, No.1, pp.18–28 (1996).
3) 位守弘充，中村 宏，朴 泰祐，中澤喜三郎：スライドウィンドウ方式による擬似ベクトルプロセッサ，情報処理学会論文誌，Vol.34, No.12, pp.2612–2623 (1993).
4) Nakamura, H., Wakabayashi, T., Nakazawa, K., Boku, T., Wada, H. and Inagami, Y.: Pseudo Vector Processor for High-Speed List Vector Computation with Hiding Memory Access Latency, *Proc. TENCON '94*, pp.338–342 (1994).
5) Callahan, D., Kennedy, K. and Porterfield, A.: Software Prefetching, *Proc. 4th Intl. Conf. on Architectual Support for Programming Languages and Operating Systems*, pp.40–52 (1991).
6) Mowry, T.C., Lam, M.S. and Gupta, A.: Design and Evaluation of a Compiler Algorithm for Prefetching, *Proc. 5th Intl. Conf on Architectual Support for Programming Languages and Operating Systems*, pp.62–73 (1992).
7) 長島重夫，田中義一：スーパーコンピュータ，オーム社 (1992).
8) Hewlett-Packard Company: PA-RISC 1.1 Architecture and Instruction Set Reference Manual (1990).
9) Wong, W.F., Oyanagi, Y. and Goto, E.: Supercomputer Performance Evaluation using Six Benchmarks, *Proc. TENCON '94*, pp.1107–1111 (1994).

(平成 8 年 3 月 26 日受付)
(平成 8 年 7 月 4 日採録)

**廣野　哲**（正会員）

昭和45年生．平成6年筑波大学第三学群情報学類卒業．平成8年同大学院理工学研究科修士課程修了．同年（株）日立製作所入社．現在に至る．計算機アーキテクチャ，並列処理に興味を持つ．

**朴　泰祐**（正会員）

昭和59年慶應義塾大学工学部電気工学科卒業．平成2年同大学院理工学研究科電気工学専攻後期博士課程修了．工学博士．昭和63年同大学理工学部物理学科助手．平成4年筑波大学電子・情報工学系講師，平成7年同助教授，現在に至る．超並列計算機アーキテクチャおよび並列処理言語・アルゴリズムの研究に従事．電子情報通信学会会員．

**中村　宏**（正会員）

昭和60年東京大学工学部電子工学科卒業．平成2年同大学院工学系研究科電気工学専攻博士課程修了．工学博士．同年筑波大学電子・情報工学系助手．同講師，同助教授を経て，平成8年より東京大学先端科学技術研究センター助教授．計算機アーキテクチャ，並列処理，計算機の上位レベル設計支援に関する研究に従事．本会平成5年度論文賞，本会平成6年度山下記念研究賞各受賞．電子情報通信学会，IEEE，ACM各会員．

**中澤喜三郎**（正会員）

昭和30年東京大学工学部応用物理卒業．昭和35年同大学院数物系博士課程応用物理修了．同年日立製作所入社．TAC，HITAC 5020，E/F，8800/8700，M-200H/280H，680H，S-810等，超大型コンピュータ・スーパーコンピュータの開発に従事．平成元年より筑波大学電子・情報工学系教授，計算物理学センター向きの超並列処理システムCP-PACSの研究に従事．平成8年より電気通信大学情報工学科教授，現在に至る．工学博士．電子情報通信学会，IEEE，ACM各会員．平成5年度本学会論文賞受賞．