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

位守弘充[†] 中村 宏[††]
朴 泰祐[††] 中澤喜三郎[††]

大規模科学技術計算では，データ領域が非常に大きくデータの局所性が少ないため，キャッシュメモリが有効に働かない．そのためスカラプロセッサの実効性能はキャッシュミス時の主記憶アクセスペナルティーにより低下する．本論文では，主記憶のスループットを十分強化した上で，浮動小数点レジスタの構成としてスライドウィンドウ方式を採用し，既存のスカラアーキテクチャとの上位互換性を保ちながらレジスタ数を増やすことでこの問題を解決した新しいプロセッサを提案する．提案するスライドウィンドウ方式は，われわれが以前提案したレジスタウィンドウ方式と比較して，ウィンドウ構成をソフトウェアで制御できるという長所がある．本論文ではスライドウィンドウを用いた擬似ベクトルプロセッサのアーキテクチャと処理原理，ならびにベンチマークプログラムを用いた評価結果を示す．主記憶アクセスレーテンシーが20マシンサイクルの場合，提案するプロセッサは通常のスカラプロセッサに対し約8倍の性能向上が得られた．レジスタウィンドウ方式のプロセッサと比べても，レジスタ数が同じ場合，2倍の主記憶アクセスレーテンシーを隠蔽でき，総レジスタ数が88のとき，提案するプロセッサは60マシンサイクルの主記憶アクセスレーテンシーを隠蔽することができた．これらの評価結果より，提案するプロセッサは高速にベクトル計算を処理することができると結論できた．

Pseudo Vector Processor Based on Slide-Windowed Registers

Hiromitsu Imori,[†] Hiroshi Nakamura,[††] Taisuke Boku[††]
and Kisaburou Nakazawa[††]

In engineering/scientific applications, caches do not work effectively. This is because there is little temporal locality and large amount of data are required in these applications. Therefore, the performance of the ordinary scalar processors is seriously degraded by the penalty of main memory access. In order to avoid this degradation, we propose a new pseudo vector processor based on *slide-windowed registers* : PVP-SW. The proposed processor can tolerate main memory access latency by introducing slide-windowed registers with preloading feature and pipelined memory. The processor holds upward compatibility with existing scalar architectures. Compared with our previous work of register window, registers can be utilized more flexibly in the proposed processor. This paper describes the architecture, the principle, and the evaluation results of the proposed processor. The evaluation results show that the proposed processor can drastically reduce the penalty of main memory access. This new processor with 88 registers can tolerate the main memory access latency of 60 CPU cycles. This is two times longer than what our previous work of register window can tolerate. From these results, we concluded that the proposed processor is very suitable for high-speed vector processing.

1. はじめに

スーパーコンピュータの1つの方向として，スカラプロセッサをノードプロセッサとする超並列処理方式が有望視されており，ベクトル処理方式より1桁以上高いピーク性能をもつ商用機が発表されている[1]．スカラプロセッサを用いた超並列スーパーコンピュータが期待される背景として，集積回路技術の進歩によるクロック周波数の向上および，スーパースカラ方式等の，命令レベルの並列性を抽出し複数個用意された演算器等を切れ目なく稼働させることで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が飛躍的に向上していることが挙げられる．

しかしながら，そのスカラプロセッサの高い処理能力は，キャッシュメモリが有効に働くときにのみ達成される．超並列スーパーコンピュータが目指している大規模科学技術計算では並列処理を行うとはいえ，1つのノードプロセッサが扱わなければならないデータ

† 日立製作所汎用コンピュータ事業部
General Purpose Computer Division, Hitachi, Ltd.
†† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

領域が非常に大きくデータの局所性が少ないという性質があるため，通常のキャッシュ，とくにデータキャッシュが有効に働かないことが多い．その場合，スカラプロセッサの性能は主記憶アクセスペナルティーにより大きく低下する．この問題は文献2)で報告されている．

スカラプロセッサとベクトルプロセッサとの大きな相違点は，主記憶のデータ供給能力にある．ベクトルプロセッサでは主記憶アクセスが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もつことによりベクトルロード/ストア処理のベクトル長を長くでき，チェイニングの機構によりこれらの処理と演算処理とを並行に行えるため，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テンシーが性能に与える影響をかなり隠蔽できる[14]．しかしながら，ワークステーション等に用いられているスカラプロセッサの主記憶はパイプライン化されていない．そのため，スループットは低く，キャッシュミス時には主記憶アクセスペナルティーにより実効性能は大きく低下する．

したがってスカラプロセッサにおいても，主記憶アクセスペナルティーを隠蔽することができれば，1つのベクトル命令の処理内容を，複数のスカラ命令によって垂直マイクロプログラミング的に処理することにより，演算器を切れ目なく稼働させることができ，高速にベクトル処理を行うことができると考えられる[15]．この手法を本論文では**擬似ベクトル処理**[3]と呼ぶ．

主記憶アクセスペナルティーを隠蔽し擬似ベクトル処理を実現するためには，先に述べた相違点である，主記憶アクセスのパイプライン化，多数のレジスタ，レジスタへのプリロード機能が必要となる．われわれはこの3点を既存のスカラアーキテクチャの拡張により実現した擬似ベクトルプロセッサ **PVP-RW** (Pseudo Vector Processor based on Register Window) を提案し，その有効性を確認している[4)~6)]．PVP-RW は，浮動小数点レジスタをレジスタウィンドウ構成にすることにより，拡張前のアーキテクチャと上位互換性を保ちながらレジスタ数を増加することが可能である．

しかし，PVP-RW にはウィンドウ構成がアーキテクチャとして固定であるため，すべてのアプリケーションに対して最適な構成になっているわけではないという問題があった[6]．

そこで本論文では浮動小数点レジスタをスライドウィンドウ構成とした擬似ベクトルプロセッサ PVP-SW (Pseudo Vector Processor based on Slide-Windowed registers) を新たに提案する．PVP-SW のスライドウィンドウ構成は，処理するアプリケーションに応じてウィンドウをソフトウェアにより制御することができる．本論文ではアーキテクチャとその処理原理を説明し，ベンチマークプログラムを用いた評価結果を示す．

2. PVP-RW のアーキテクチャとその処理原理

本章では，PVP-RW のアーキテクチャと処理原理について簡単に述べる．詳細は文献4)～6)を参照されたい．

2.1 アーキテクチャ

PVP-RW のアーキテクチャは既存のスカラアーキテクチャの拡張として定義でき，拡張前のアーキテクチャとは上位互換性を保てる．主な拡張点は以下の2点である．

(1) レジスタウィンドウ構成：

浮動小数点レジスタはレジスタウィンドウ構成となる．**図1**はレジスタウィンドウの構成例である．図1のウィンドウ構成例では総レジスタ数は88であり，論理的に4つのウィンドウに分割されている．1ウィンドウのレジスタ数は拡張前のアーキテクチャのレジ

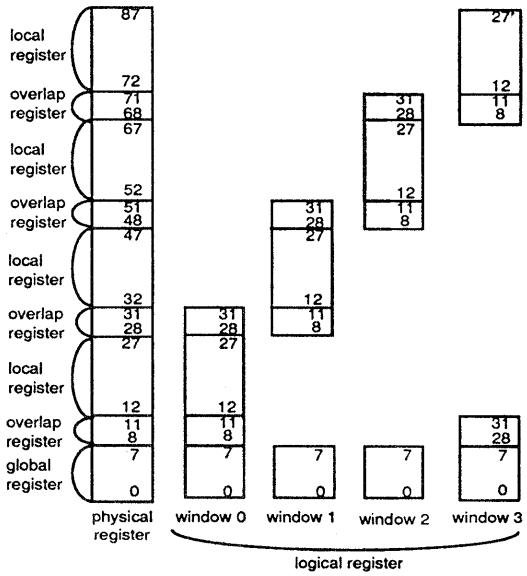


図1 レジスタウィンドウ構成
Fig. 1 Structure of register window.

スタ数と同じである．ある時刻において，active なウィンドウは1つであり，次に述べる追加命令以外の命令は active ウィンドウ内のレジスタのみを使用する．これにより拡張前のアーキテクチャと上位互換性を保てる．

（2） 追加命令：

- ウィンドウ利用可否命令：レジスタウィンドウの利用可否を指定する特権命令．
- ウィンドウ切り替え命令：active なウィンドウを変更する．
- プリロード命令：active なウィンドウから1つ先のウィンドウの論理レジスタを指定してデータをプリロードする．
- 拡張プリロード命令：active なウィンドウから2つ先のウィンドウの論理レジスタを指定してデータをプリロードする．
- ポストストア命令：active なウィンドウから1つ前のウィンドウの論理レジスタを指定してストアを行う．

2.2 処理原理

科学技術計算において最も処理能力を必要とするのはループ処理である．PVP-RW は主記憶アクセスレーテンシーを隠すことにより高速にループ処理を実行できる．

図2にその処理原理を示す．1つのループ処理は，概念的にデータのロード/演算処理/結果のストアの3つの phase から成る．PVP-RW ではこれらを，preload/execution/poststore の3つの phase に分割し，各 phase を異なるレジスタ空間上でパイプライン的に処理する（**phase pipelining**）[5]．例えば図2において，(i)-th iteration を実行中の active window は window j であるが，この iteration において以下の処理が行われる．

- preload phase：次の ($i+1$)-th iteration で用いられるデータをプリロード命令により1つ先のウィンドウ（window $j+1$）にプリロードする．
- execution phase：(i)-th iteration の実行を現在 active であるウィンドウ（window j）で処理する．
- poststore phase：($i-1$)-th iteration で計算されたデータをポストストア命令を用いて1つ前のウィンドウ（window $j-1$）からポストストアする．

いま，図2において時刻 Tld にあるデータのプリロード命令が発行され，そのデータが時刻 Tex で実際に計算に使用されるとする．この Tld と Tex との時間間隔が主記憶アクセスレーテンシーより長ければ，主記憶アクセスによる性能低下は防げる．

phase pipelining では各 phase がそれぞれ1つのウィンドウのレジスタを占有できるため，十分な loop unrolling を行うことができる．これにより各 phase の長さを長くでき，Tld と Tex の時間間隔を長くできる．このようにして PVP-RW は主記憶アクセスレーテンシーを隠蔽できる．

また，2つ先のウィンドウへのプリロード命令を用いれば，さらに長い主記憶アクセスレーテンシーを隠蔽することができる．

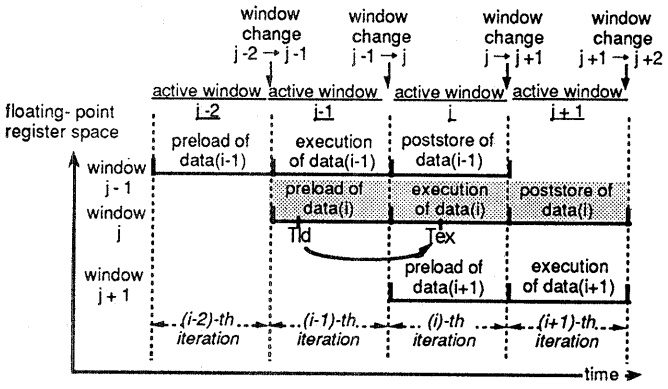


図2 phase pipelining による擬似ベクトル処理
Fig. 2 Pseudo vector processing in phase pipelining.

3. スライドウィンドウを用いた擬似ベクトル処理

本章では本論文で新たに提案する，浮動小数点レジスタをスライドウィンドウ構成とした擬似ベクトルプロセッサ（Pseudo Vector Processor based on Slide-Windowed registers：PVP-SW）とその処理原理について説明する．PVP-SW のアーキテクチャは，既存のスカラアーキテクチャの拡張として定義でき，拡張前のアーキテクチャとは上位互換性を保てる．

3.1 PVP-SW のアーキテクチャ

3.1.1 浮動小数点レジスタのスライドウィンドウ構成

ウィンドウ構成をソフトウェアで指定できるようにレジスタウィンドウを拡張したものを**スライドウィンドウ**と呼ぶ．PVP-SW のスライドウィンドウ構成例

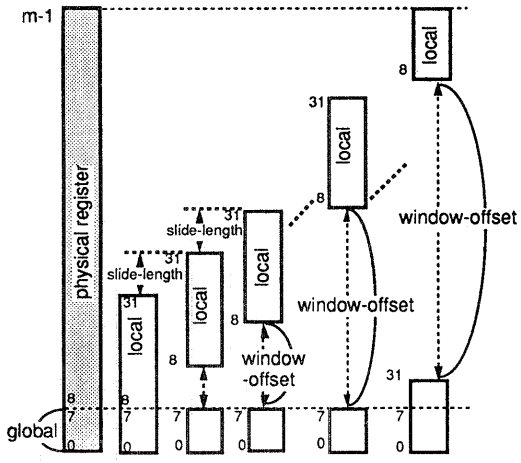


図 3 スライドウィンドウ構成
Fig. 3 Structure of slide-windowed registers.

を図3に示す．ここでは浮動小数点レジスタの総数を m とする．m 個の物理レジスタは論理的に複数のウィンドウに分割され，各ウィンドウの位置は **window-offset** により与えられる．

PVP-RW との類似点は，ある時刻において active であるウィンドウは1つである，1ウィンドウ内のレジスタ数は拡張前のアーキテクチャのレジスタ数と同じである，の2点である．

- **Slide-Pitch**：ウィンドウ位置の指定単位を**slide-pitch** とする．例えば slide-pitch が2の場合，総レジスタ数 m から global register 数8を除いた $(m-8)/2$ 個のウィンドウを指定可能になる．総レジスタ数 m および slide-pitch は，実際にはアーキテクチャにより一意に定義される値であるが，以降一般的に m，slide-pitch は任意の値を取り得るものとして説明する．
- **FWSTP, FWSTPE**：現在どのウィンドウが active であるかを示す情報を保持する **FWSTP** (Floating Window STart Pointer を floating-point status register 内に設ける．$(m-8)$/slide-pitch 個のウィンドウを指定するのに必要な FWSTP の bit 数を n とすると，n は(3.1)式により与えられる．

$$n=\left\lceil \log_2 \frac{m-8}{\text{slide-pitch}} \right\rceil \qquad (3.1)$$

$\lceil a \rceil$：切り上げを表す．

スライドウィンドウ機構を用いるか否かを示す1bit の **FWSTPE** (Floating Window STart Pointer Enable)を **PSW** (Program Status Word) 内に設ける．FWSTPE=0 としてスライドウィンドウ機構を使用しない場合，1つのウィンドウのレジスタのみを使用することになり，拡張前のアーキテクチャと完全互換性を保てる．

- **Active なウィンドウの Window-Offset**：active なウィンドウの window-offset は(3.2)式により決定される．

$$\text{window-offset}=\begin{cases}\text{if FWSTPE}=0 \text{ then } 0\\ \text{if FWSTPE}=1 \text{ then slide-pitch}\times\text{FWSTP}\end{cases} \qquad (3.2)$$

- **物理レジスタ番号と論理レジスタ番号との対応**：次節で述べる追加命令以外の命令は，active なウィンドウ内のレジスタのみを使用し，そのレジスタ指定はすべて論理レジスタ番号により行われる．したがって論理レジスタ番号から物理レジスタ番号への変換が必要になる．物理レジスタ番号 #Rは，論理レジスタ番号 #r と window-offset により以下のように決定される．

$$\begin{cases}\text{if } 0\leq\#r\leq 7\text{(global) then } \#R=\#r\\ \text{if } 8\leq\#r\leq 31\text{(local)}\\ \qquad\text{then } \#R=\{(\text{window-offset}\\ \qquad\qquad +(\#r-8))\bmod(m-8)\}+8\end{cases} \qquad (3.3)$$

- **Slide-Length と Window-Stride**：active なウィンドウを切り替えるときの，現在 active なウィンドウと，次に active になるウィンドウとの間の距離を **slide-length** で表す．slide-length は(3.4)式により決定される．

$$\text{slide-length}=\text{slide-pitch}\times\text{window-stride} \qquad (3.4)$$

(3.4)式の **window-stride** は，ウィンドウ切り替え命令である FWSTPset 命令 (3.1.2項参照) により与えられる．

3.1.2 追加命令

PVP-SW において追加する命令は以下の4つである．

- **FWSTPenable**：FWSTPE を設定し，スライドウィンドウ機構を用いるか否かを決定する特権命令．
- **FWSTPset**：FWSTP の値をセットする非特権命令．この命令は 1bit の s/i field(set/increment field) と n bit の window-stride field をも

以下のように FWSTPbit を設定する．

```
if s/i field=0(set) then
        FWSTP:=window-stride
if s/i field=1(increment) then
    FWSTP:=(FWSTP+window-stride)mod
        {(m−8)/slide-pitch}
```

(3.5)

ち，s/i=0 のとき，window-stride の値を直接 FWSTP に代入することで，物理レジスタ空間におけるウィンドウの位置を絶対的に決定することができる．また s/i=1 のときは，FWSTP を window-stride の値によりインクリメントし，相対的にウィンドウを切り替えていく．

- **FRPreload**：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．5 bit の論理レジスタ指定 field と n bit の window-stride field をもち，ロード先のウィンドウの window-offset は(3.6)式により与えられる．ロード先の物理レジスタ番号はこの window-offset と命令により指定される論理レジスタ番号から(3.3)式によって導出される．

$$\text{window-offset(destination window)} = \{(\text{FWSTP}+\text{window-stride}) \times \text{slide-pitch}\} \bmod (m-8) \quad (3.6)$$

動作は通常のロード命令と同様であるが，キャッシュミス時には，キャッシュへのブロック転送は行わず，主記憶よりレジスタへデータを直接転送する．したがってキャッシュ内のデータは更新されない．

- **FRPostore**：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．キャッシュとの関係，および 5 bit の論理レジスタレジスタ指定 field と n bit の window-stride field が命令中にある点は FRPreload 命令と同様である．ストア元のウィンドウの window-offset は(3.7)式によって与えられる．

$$\text{window-offset(source window)} = \{(\text{FWSTP}-\text{window-stride}) \times \text{slide-pitch}\} \bmod (m-8) \quad (3.7)$$

3.1.3 PVP-RW との関係

PVP-RW のアーキテクチャは PVP-SW に包含される．図1のウィンドウ構成を取る PVP-RW は，$m=88$, slide-pitch=20 である PVP-SW と等価となる．

PVP-SW と PVP-RW の相違点は以下の2つである．第1に PVP-SW ではウィンドウ構成，すなわち active なウィンドウを切り替えるときのウィンドウ間距離をソフトウェアにより制御できることである．PVP-RW では，ウィンドウ切り替え時には必ず1つ先のウィンドウが active になるため，(3.4)式の slide-length にあたるものがアーキテクチャとして固定されたものとなる一方，PVP-SW では slide-length をソフトウェアで指定できるため任意のウィンドウ構成を選択できる．第2に PVP-SW では任意の指定ウィンドウにデータをプリロードできることである．PVP-RW のプリロード命令，拡張プリロード命令は，PVP-SW における FRPreload 命令の window-stride を1と2に固定したものとみなすことができる．PVP-SW の FRPreload 命令は，window-stride を命令中で任意に指定できる．

3.2 PVP-SW における擬似ベクトル処理の原理

3.2.1 擬似ベクトル処理例

PVP-SW における擬似ベクトル処理の原理を説明するため，例として内積演算 (c=c+a(i)×b(i)) を取り上げる．

仮定として FRPreload 命令の主記憶アクセスレーテンシーを7マシンサイクル (MC)，浮動小数点演算命令のレーテンシーを 3 MC とする．説明を簡単にするため分岐命令は省略する．

slide-pitch を2としたときの1ループのオブジェクトコードを**図4**に示す．オブジェクトコード中の 'FRPreload r30〈+2〉' は，2つ先のウィンドウの論理レジスタ #r30 にデータをプリロードすることを意味する．ウィンドウはループごとに FWSTPset 命令により切り替わる．図4のオブジェクトコードでは，各ループごとに新たに必要となる2つのデータを新しい物理レジスタに割り当てる必要がある．したがってウィンドウ切り替え時の slide-length は2以上である

```
Loop: ADD       r4〈-r4+r29      %r4(c)〈-r4(c)+r29(a(i−1)×b(i−1))[window j]
      MULT      r31〈-r31×r30    %r31(a(i)×b(i))〈-r31(a(i))×r30(b(i))[window j]
      FRPreload r31〈+2〉         %r31〈+2〉[window j+2]〈-a(i+2)
      FRPreload r30〈+2〉         %r30〈+2〉[window j+2]〈-b(i+2)
      FWSTPset  +1               %(FWSTPset〈-FWSTPset+1)[FWSTP: j〈-j+1]
```

図4 内積演算 (c=c+a(i)×b(i)) のオブジェクトコード
Fig. 4 Object code of inner product (c=c+a(i)×b(i)).

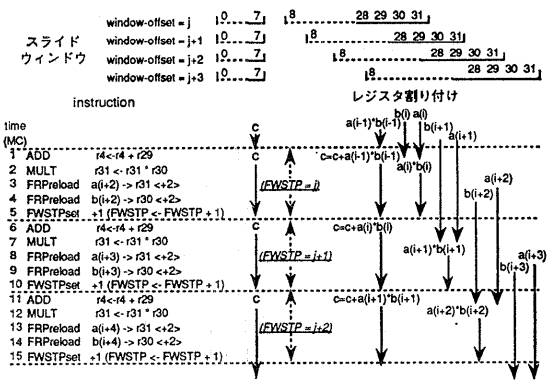


図 5 PVP-SW における内積計算のレジスタ割り付け
Fig. 5 Register allocation of inner product on PVP-SW.

必要がある．いま，slide-pitchが2であるので，(3.4)式より FWSTPset 命令の window-stride は1以上となる．ここでは，window-stride を1とし，'FWSTPset+1' で表す．

図4の右側の記述は，active なウィンドウが window j であるときの各命令の動作を示す．[ ] 内は，各命令が使用するウィンドウを表している．

図4のオブジェクトコードを実行するときのレジスタ利用の様子を**図5**に示す．5MC ごとの点線は，FWSTPset 命令によりウィンドウが切り替わっている様を示す．

各ループの FRPreload 命令は，2つ先のウィンドウのレジスタにデータをあらかじめプリロードする．演算命令は現在 active なウィンドウのレジスタのみを用いて処理を行う．次のウィンドウとオーバーラップしているレジスタは，ウィンドウ間のデータの引渡しに用いられる．

ループごとにウィンドウを切り替えるため，各ループ処理は異なるウィンドウ上で実現される．すなわち同一の論理番号をもつレジスタも，ループごとに物理的には異なるレジスタを指定していることになる．これにより 32 レジスタしか指定できない演算命令が多数のレジスタを用いて処理を行うことができる[7]．

3.2.2 Permitted Latency

プリロード命令によるロード要求と，演算命令による使用との時間間隔の最小値を **permitted latency** と定義する[5]．permitted latency は与えられたオブジェクトコードにより一意に決まる．permitted latency が主記憶アクセスレーテンシーより長ければ主記憶アクセスペナルティーによる性能低下は起こらない．したがって permitted latency を長くすることが重要になる．図5の例では，3MC 目で発行された a(i+2) と 4MC 目で発行された b(i+2) が，12MC 目の演算で用いられるので，b(i+2) に対する時間間隔である 8MC が，図5のオブジェクトコードにおける permitted latency となる．

一般に permitted latency は(3.8)式により与えられる．

$$\text{permitted latency} = \text{loop-cycle} \times \left\lfloor \frac{\text{window-stride(FRPreload)}}{\text{window-stride(FWSTPset)}} \right\rfloor + \delta \tag{3.8}$$

$\lfloor\alpha\rfloor$: 切り捨てを表す．

ただし，loop-cycle : 1ループの実行時間 (MC)．

window-stride (FWSTPset) : FWSTPset 命令での window-stride.

window-stride (FRPreload) : FRPreload 命令での window-stride.

δ = window-stride (FRPreload) が0のときの permitted latency.

図4のオブジェクトコードにおいて，window-stride (FRPreload) を0とすると，4番目の FRPreload 命令でロードされる配列 b のデータが，2番目の乗算命令で使用されることになる．したがって $\delta=-2$ となる．このとき，図4のオブジェクトコードの permitted latency は，(3.8)式に loop-cycle=5, window-stride (FWSTPset)=1, window-stride (FRPreload)=2, $\delta=-2$ を代入することにより求められる．

3.2.3 Permitted Latency の最大値

(3.8)式において，loop-cycle, window-stride (FWSTPset), δ は計算対象により一意に決まる．したがって permitted latency が最大値をとるのは，最も遠いウィンドウにプリロードを行うときであり，これは window-stride (FRPreload) の値を最大にするときである．プリロード先のレジスタは，演算に使用中のレジスタとの重複を避ける必要があるため，window-stride (FRPreload) は以下の式を満たす必要がある．

$$\text{window-stride(FRPreload)} \leq \left\lfloor \frac{m-8-(\#\text{ of utilized registers})}{\text{slide-pitch}} \right\rfloor \tag{3.9}$$

# of utilized registers : 1ループで使用されている global register 以外の論理レジスタ数を表す．

(3.9)式により与えられる window-stride (FRPreload) の最大値を(3.8)式に代入することにより，permitted latency の最大値が得られる．

図4のオブジェクトコードにおいては #r 29〜#r 31 を演算に使用しているので，# of utilized register は3である．したがって $m=64$ ならば，(3.9)式より window-stride (FRPreload) の最大値は26となり，(3.8)式に代入することにより permitted latency の最大値 128 MC が得られる．

3.3 コンパイル手法

PVP-RW では，ウィンドウ構成が固定であるため，コンパイラはこの構造を意識しながら loop unrolling などの最適化を行う必要があった．それに対し，PVP-SW ではコンパイラが slide-length を任意に決定できるため，計算対象の性質に合わせて自由にレジスタを活用できる．

コンパイル手法の詳細は別の論文[16]に譲るが，PVP-SW 上での最適コードはすでに提案されている手法を応用することで生成可能であることを確認している．すなわち命令スケジューリングに関しては，software pipelining の1つである modulo scheduling[8]を用いることができ，レジスタ割り付けに関しても，「ウィンドウ間にデータを引き渡す際に，そのデータに割り付けられる論理レジスタ番号は，slide-length の差がなければならない」という制約条件が必要になるものの，この制約条件を満たせば register coloring[9]を用いることができる．

4. 評価手法

4.1 評価モデル

すでに述べたように，擬似ベクトルプロセッサは既存のスカラアーキテクチャの拡張により実現可能である．本論文では拡張するスカラアーキテクチャの1例として，下記にその要点を示した Hewlett-Packard 社の PA-RISC 1.1 Architecture[10]を取り上げて評価を行った．

- ロード/ストアアーキテクチャであり，4もしくは8byte のデータがロード/ストア命令によって転送される．
- 命令長は 4 byte 固定である．
- 汎用/浮動小数点レジスタは両方とも32個で構成され，浮動小数点レジスタは1レジスタ 8 byte 長である．ただし，浮動小数点レジスタに関しては #r 0〜#r 3 が予約されており，ユーザに開放されているのは残り28レジスタのみである．
- 1つの複合命令で2種類の演算を指定できる命令があり，たとえば，'FMPYADD *rm* 1, *rm* 2, *tm*, *ra*, *ta*' という1つの命令は，''FR[*tm*]〈-FR[*rm* 1]×FR[*rm* 2]; FR[*ta*]〈-FR[*ta*]+FR[*ra*]'' という乗算と加算の両方を行うことができる．ただし $ra \neq tm$, $ta \neq rm1$, $rm2$, tm という制約条件が付く．

PVP-SW の有効性を検証するため，以下の4つのプロセッサモデルに対し評価を行った．各モデルの概要を以下に示す．**表1**に各モデルの特徴をまとめる．

〈original〉: 拡張を行わない PA-RISC 1.1 Architecture を採用するモデルであり，主記憶はパイプライン化されていない．

〈PVP-RW-88〉: 図1に示すレジスタウィンドウを導入したモデル．主記憶のパイプライン化と，プリロード/ポストストア命令のスカラアーキテクチャへの追加を行っている．

〈PVP-SW-*m*/slide-pitch〉: 本論文で提案するスライドウィンドウを用いて擬似ベクトル処理を行うモデル．主記憶のパイプライン化と，プリロード/ポストストア命令のスカラアーキテクチャへの追加を行っている．物理的な浮動小数点レジスタの総数を *m* とし，48, 64, 80, 88, 96, 112, 128 を仮定する．slide-pitch として1, 2, 4, 8を仮定する．

〈ideal〉: アーキテクチャは〈original〉と同じだが，必要となるデータすべてがデータキャッシュに納まっていることを仮定した理想的なモデル．

表1 評価プロセッサモデルの仕様
Table 1 Specification of evaluated processor model.

評価モデル	Architecture	主記憶	Data cache	Preload/prefetch の実現方法	レジスタ構造	レジスタ数
Original	PA-RISC 1.1	not pipelined	conventional fully associative	なし	拡張なし	32
PVP-RW-88	拡張 PA-RISC 1.1	pipelined	conventional	register への preload	register window	88
PVP-SW	拡張 PA-RISC 1.1	pipelined	conventional	register への preload	slide window	*m* (48〜128)
Ideal	PA-RISC 1.1	not pipelined	∞ all cache hit	なし	拡張なし	32

4.2 評価条件

評価条件を以下に記す．

- 命令発行：2命令発行のスーパースカラ方式とする．ストールが発生すると，後続命令はインタロックされる．同時発行される2命令は，ロード/ストア系（FRPreload 等），浮動小数点演算，整数演算（FWSTPset を含む），その他（分岐命令）のいずれか異なるグループに属さなければならない．
- データ依存関係：先行命令が浮動小数点演算なら5MC 以上，先行命令がロード命令ならキャッシュヒット時に2MC 以上，キャッシュミス時なら主記憶アクセスレーテンシー以上，後続命令の発行を遅らせるとする．
- データサイズは 8 byte とする．主記憶のスループットは 8 byte/cycle とし，バンクコンフリクトは生じないものと仮定する．
- データキャッシュ：ブロックサイズ 16 byte とし，〈original〉では line conflict は起こらないとする．
- 命令キャッシュ：必要な命令は命令キャッシュ内に格納されているものとする．

4.3 評価用ベンチマークプログラム

Livermore Fortran Kernels (LFK) を用いて評価を行う．ただし実際の科学技術計算のアプリケーションに近付けるため，ループの繰り返し回数を増加させており，ループ処理の定常状態を評価対象とする．オブジェクトコードはハンドコンパイルにより各モデルごとに最適化し，〈PVP-SW〉に関しては 3.3 節で触れたアルゴリズムに沿って生成している．評価は命令パイプラインのステージレベルのシミュレーションにより行った．

5. 評価と考察

5.1 評価結果

（1） 主記憶アクセスレーテンシーが 20 MC の場合

主記憶アクセスレーテンシーが 20 MC のときの各プロセッサモデルの実効性能を**図 6** に示す．性能は FLOPC (FLoating-point Operations Per Clock Cycle) で表す．PA-RISC 1.1 Architecture では複合命令により1命令で2つの浮動小数点演算を処理でき，1MC で1つの浮動小数点演算命令を発行可能と仮定しているので理論ピーク性能は 2 FLOPC となる．ここでは〈PVP-SW〉のモデルとして，総レジスタ数 $m=88$, slide-pitch=2 である〈PVP-SW-88/2〉を取り上げている．

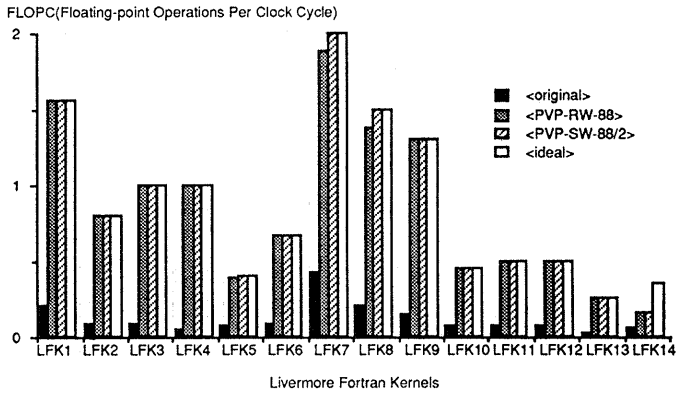


図 6 主記憶アクセスレーテンシーが 20 MC のときの各プロセッサモデルの性能
Fig. 6 Performance of each processor model (memory access latency=20 MC).

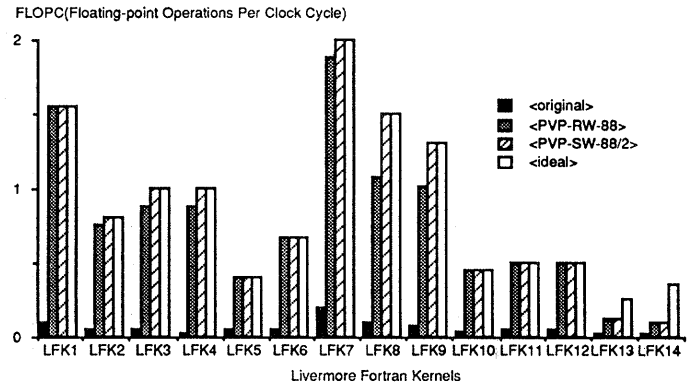


図 7 主記憶アクセスレーテンシーが 50 MC のときの各プロセッサモデルの性能
Fig. 7 Performance of each processor model (memory access latency=50 MC).

〈original〉はキャッシュミス時の主記憶アクセスペナルティーにより大きく性能が低下し，〈ideal〉と比較してほぼ 5〜20% の性能しか達成していない．一方，〈PVP-RW-88〉は〈ideal〉の 45%〜100% の性能を維持している．LFK #14 を除けば，〈ideal〉の 92%〜100% であり，主記憶アクセスレーテンシーの影響をほぼ完全に隠すことができていることがわかる．〈PVP-SW-88/2〉は，LFK #14 を除けば〈ideal〉と性能がまったく変わらず，主記憶アクセスレーテンシーの影響を完全に隠蔽していることがわかる．〈PVP-SW-88/2〉は〈original〉に対し約8倍の性能向上が得られている．〈PVP-RW-88〉は〈PVP-SW-88/2〉と比較して，LFK #5, LFK #7, LFK #8 でわずかに性能が低い．これは図1に示した〈PVP-RW_88〉のウィンドウ構成では，次のウィンドウに引き渡せるデータ数が global register と overlap register のレジスタ数である 12 以下に制限されるため，オブジェクトコードの命令スケジューリングが制限され，命令ステップ数が増加するためである．このことは，これらの LFK に対して図1のウィンドウ構成が最適でないことを表している．〈PVP-SW-88/2〉は各 LFK ごとに最適なウィンドウ構成を選択できるため，〈PVP-RW-88〉よりも高い実効性能が得られている．

（2） 主記憶アクセスレーテンシーが 50MC の場合

次に主記憶アクセスレーテンシーを 50 MC と仮定した場合の結果を**図7**に示す．

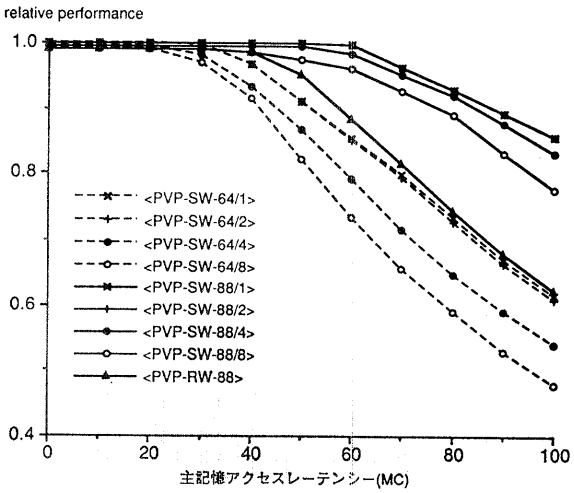


図 8 主記憶アクセスレーテンシーを変化させたときの各〈PVP-SW〉モデルの性能

Fig. 8 Relative performance of each 〈PVP-SW〉 Model under variable memory access latency.

（1）の場合よりキャッシュミス時のペナルティーが大きいため，〈original〉は〈ideal〉のほぼ 2〜10% の性能にまで低下している．〈PVP-RW-88〉と〈PVP-SW-88/2〉を比較すると，〈PVP-RW-88〉では LFK #2, LFK #3, LFK #8 等いくつかの LFK において主記憶アクセスレーテンシーを隠し切れず実効性能が低下しているのに対し，〈PVP-SW-88/2〉では完全にその影響を隠蔽できていることがわかる．

（3） 主記憶アクセスレーテンシーを変化させた場合

主記憶アクセスレーテンシーを変化させたときの〈PVP-RW-88〉，〈PVP-SW〉の性能を**図8**に示す．図8では各モデルの実効性能を〈ideal〉の実効性能を1としたときの相対性能値で表している．相対性能はベクトルプロセッサでベクトル化できる LFK #1〜LFK #12 の性能の調和平均より求めている．〈PVP-SW〉のモデルとして m=64, 88 と slide-pitch=1, 2, 4, 8 を組み合わせた8通りを取り上げる．

図8よりわかるように，〈PVP-RW-88〉は主記憶アクセスレーテンシーが 30 MC あたりから性能低下が起こり始めるのに対し，〈PVP-SW-88/2〉は主記憶アクセスレーテンシーが 60 MC あたりまで実効性能が維持されている．したがって総レジスタ数が同じであるにもかかわらず，〈PVP-SW-88/2〉は〈PVP-RW-88〉の約2倍の主記憶アクセスレーテンシーを隠すことができている．また，総レジスタ数 64 の〈PVP-SW-64/2〉は，〈PVP-RW-88〉より総レジスタ数が24少ないにもかかわらず，主記憶アクセスレーテンシーが 30 MC まではまったく性能低下が起こらず，〈PVP-RW-88〉と同程度の実効性能である．

次に〈PVP-SW〉の各モデルについて検討する．図8において主記憶アクセスレーテンシーが 20 MC のとき，〈PVP-SW-64/4〉，〈PVP-SW-64/8〉の実効性能が〈PVP-SW-64/1〉，〈PVP-SW-64/2〉より低下しているのは，slide-pitch の値が大きくなることで，ウィンドウ構成の選択の自由度が低下し，ウィンドウ構成を最適にできないためである．主記憶アクセスレーテンシーが長い場合をみると，総レジスタ数が多いほうが，また総レジスタ数が一定のときには slide-pitch が小さいほうが主記憶アクセスペナルティを効率よく隠蔽していることもわかる．

（1），（2），（3）の結果から主記憶アクセスペナルティの実効性能への影響を軽減させ，高い

実効性能を達成する手法として，〈PVP-SW〉は〈PVP-RW〉より優れており，〈PVP-SW〉においても総レジスタ数は大きく，slide-pitch は小さいほうが優れていることがわかる．

5.2 PVP-SWにおける総レジスタ数，slide-pitchの最適値の考察

5.2.1 slide-pitch の最適値

前節で述べたように slide-pitch が小さいほうが主記憶アクセスペナルティーの影響を有効に隠すことができる．しかしながら，slide-pitch を小さくすれば，FWSTP の bit 数が増加し，floating-point status register 内に保持する情報量が増えるという問題点がある．これらを考慮すると，slide-pitch の値としては1または2が望ましいといえる．

5.2.2 総レジスタ数 m の最適値

次に総レジスタ数 m の最適値に関して検討する．図8の結果が示すように，総レジスタ数 m が大きいほうが，より長い主記憶アクセスレーテンシーを隠すことができる．これはすなわち permitted latency が m につれて増加していることを表している．m と permitted latency の関係は(3.4)式，(3.9)式を(3.8)式に代入して得られる以下の近似式で与えられる．

$$\text{permitted latency} = \text{loop-cycle} \times \frac{m-8-(\#\text{ of utilized registers})}{\text{slide-length}} + \delta \qquad (5.1)$$

(5.1)式において slide-length，loop-cycle ともに各ベンチマークプログラムに依存する値であるので，loop-cycle/slide-length を const とすると(5.2)式となる．

$$\text{permitted latency} = \text{const} \times \{m-8-(\#\text{ of utilizedregisters})\} + \delta \qquad (5.2)$$

$$\text{const} = \frac{\text{loop-cycle}}{\text{slide-length}}.$$

したがって permitted latency は $O(m)$ で増加することがわかる．

図9に slide-pitch を2と固定し，主記憶アクセスレーテンシーをパラメータとして与えたときの総レジスタ数 m と実効性能の関係を示す．図9では〈ideal〉の実効性能を1としたときの相対性能値で表し，m は48～128まで変化させている．図9より，主記憶アクセスレーテンシー（0 MC～100 MC）の実効性能への影響を隠蔽するには，ほぼどれだけの物理レジスタ数

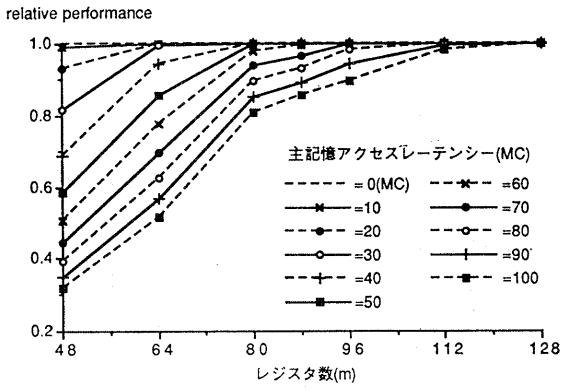


図9 総レジスタ数（m）を変化させたときの〈PVP-SW〉の性能（slide-pitch=2）

Fig. 9 Relative performance of each 〈PVP-SW〉 model under variable number of registers (m).

が必要であるか読み取ることができる．ここで仮に60 MC 程度の主記憶アクセスレーテンシーを隠蔽することを目標に設定する．

図9からわかるように，60 MC 程度の主記憶アクセスレーテンシーを隠蔽するには総レジスタ数は88～96程度で十分なことがわかる．

5.3 キャッシュへのプリフェッチとの比較

主記憶アクセスペナルティーによるスカラプロセッサの性能低下を防ぐため，あらかじめ必要なデータをキャッシュにプリフェッチする研究が報告されている[11)~13)]．しかしながら，キャッシュへのプリフェッチには以下の問題点がある．

- cache pollution：将来必要になるデータをプリフェッチする際に，現在必要なデータをキャッシュから追い出してしまう場合がある．キャッシュがダイレクトマップ方式の場合，この危険性は高くなる．
- キャッシュのスループット：データに時間的再利用性がない場合，データのプリフェッチによる主記憶/キャッシュ間のデータ転送と，キャッシュ/プロセッサ間のデータ転送を考慮すると，キャッシュに要求されるスループットは主記憶に要求されるスループットの2倍になる．したがってキャッシュのスループットを強化する必要がある．
- 不要なデータのプリフェッチ：非連続なアドレスをアクセスする場合，同一ライン内の不要なデータもプリフェッチされる．したがってキャッシュ/主記憶間で不要なデータトラフィックが生じることになる．

これに対し，提案する PVP-SW では，キャッシュを介さずにデータを主記憶から直接レジスタへ転送するため，これらの問題点は生じない．

5.4 実現可能性

PVP-SW において追加すべきハードウェアとして，数十のレジスタ，論理レジスタ番号から物理レジスタ番号への変換回路，および物理レジスタ間のデータ依存関係を処理するための制御論理の拡張が考えられる．また，1 bit の FWSTPE を PSW 内に保持し，n bit の FWSTP を floating-point status register に格納する必要がある．

実装面積からいえば，浮動小数点レジスタ数の増加のコストが大きいが，5.2.2 項で述べたように評価モデル〈PVP-SW〉では，総レジスタ数 96 レジスタ程度で非常に高い効果が得られており，実装技術の進歩を考えるとさほど難しいことはないといえる．

また，論理レジスタ番号から物理レジスタ番号に変更する回路がマシンサイクルに与える影響も少ないと考えられる．なぜならば，(3.3)式の第2式は(5.3)式のように変換できるからである．mはアーキテクチャとして決定される値であり，$(m-8)$ は定数であると考えられる．したがって，この変換に要する時間は数 bit の3入力加算器1つとセレクタ1つのみを通るパスによって決まることが(5.3)式よりわかる．

$$\begin{cases} \#R = \text{window-offset} + \#r \\ \qquad\qquad \text{if window-offset} + \#r \leq m \\ \#R = \text{window-offset} + \#r - (m-8) \\ \qquad\qquad \text{if window-offset} + \#r > m \end{cases} \tag{5.3}$$

マルチバンク構成によるパイプラインメモリは，通常のスカラプロセッサの主記憶と比較すると，コストがかかる．しかしながら，キャッシュの有効性が低下するベクトル処理では主記憶のスループットを向上させることは本質的に必要不可欠である．

6. おわりに

レジスタウィンドウ構成をソフトウェアにより制御するスライドウィンドウを用いてスカラ命令により高速にベクトル処理を行う新しいアーキテクチャ PVP-SW を提案した．

ベンチマークプログラムを用いた評価より，主記憶アクセスレーテンシーが 20 MC のとき，総レジスタ数 88，slide-pitch=2 とした PVP-SW の評価モデル〈PVP-SW-88/2〉は拡張前のスカラプロセッサに対し約8倍の性能向上が得られた．

また，〈PVP-SW〉と〈PVP-RW〉を比較すると，〈PVP-SW-88/2〉は総レジスタ数が同じである〈PVP-RW-88〉の約2倍の主記憶アクセスレーテンシーを隠すことができた．

PVP-SW において，slide-pitch は1または2が望ましく，総レジスタ数mは隠すべき主記憶アクセスレーテンシーに対してほぼ線形に増加するものの，slide-pitch が2のときで，60 MC 程度の主記憶アクセスレーテンシーを隠すためには総レジスタ数は 88～96 程度で十分なことがわかった．

以上の結果より，PVP-SW は最も効果的に主記憶アクセスペナルティーを隠しながら，ベクトル処理を高速に行えると結論できる．

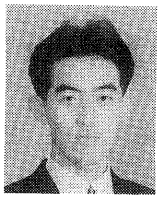
謝辞 本研究に対し有益なご意見を頂戴した筑波大学 中田育男教授，山下義行 講師，西川博昭助教授，神奈川大学 後藤英一教授，東京大学 小柳義夫教授，ならびに GNOH グループ，CP-PACS プロジェクトの方々に深く感謝いたします．なお，本研究は一部文部省科学研究費補助金（創成的基礎研究費）（課題番号 05 NP 0601）による．

参考文献

1) *CM-5 Technical Summary*, Thinking Machines Corp., Cambridge, MA. (1992).
2) Simmons, M. L. and Wasserman, H. J.: Performance Evaluation of the IBM RISC System/6000: Comparison of an Optimized Scalar with Two Vector Processors, *Proc. of Supercomputing '90*, pp. 132-141 (1990).
3) 位守弘充，伊藤元久，中村 宏，中沢喜三郎：擬似ベクトル処理向きメモリアーキテクチャの一提案，情報処理学会研究報告，ARC-91-8 (1991).
4) Nakazawa, K., Nakamura, H., Imori, H. and Kawabe, S.: Pseudo Vector Processor Based on Register-Windowed Superscalar Pipeline, *Proc. Supercomputing '92*, pp. 642-651 (1992).
5) 中村 宏，位守弘充，伊藤元久，中澤喜三郎：レジスタウィンドウとスーパースカラ方式による擬似ベクトルプロセッサの提案，並列処理シンポジウム JSPP '92 論文集，pp. 367-374 (1992).
6) 中村 宏，位守弘充，中澤喜三郎：レジスタウィンドウ方式を用いた擬似ベクトルプロセッサの評価，情報処理学会論文誌，Vol. 34, No. 4, pp. 669-680 (1993).
7) Rau, B. R., Lee, M., Tirumalai, P. P. and Schlansker, M. S.: Register Allocation for Software Pipelined Loops, *Proc. ACM SIGPLAN '92 Conf. on Programming Lan-*

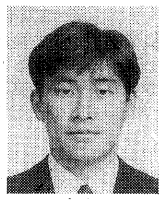
guage Design and Implementation, pp. 283-299 (1992).
8) Lam, M.: Software Pipelining: An Effective Scheduling Technique for VLIW Machines, *Proc. ACM SIGPLAN '88 Conf. on Programming Language Design and Implementation*, pp. 318-327 (1988).
9) Chaitin, G.: Register Allocation and Spilling Via Graph Coloring, *Proc. SIGPLAN Symp. on Compiler Construction*, pp. 98-105 (1982).
10) Hewlett-Packard Company: PA-RISC 1.1 Architecture and Instruction Set Reference Manual, Manual Part Number 09740-90039 (1990).
11) Jouppi, N. P.: Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers, *Proc. of 17th Int'l Symp. on Computer Architecture*, pp. 364-373 (1990).
12) Callahan, D., Kennedy, K. and Porterfield, A.: Software Prefetching, *Proc. 4th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems* (*ASPLOS-IV*), pp. 40-52 (1991).
13) Klaiber, A. C. and Levy, H. M.: An Architecture for Software-Controlled Data Prefetching, *Proc. 18th Int'l Symp. on Computer Architecture*, pp. 43-53 (1991).
14) 橋本 隆, 岡崎恵三, 弘中哲夫, 村上和彰, 富田真治:「順風」: MSF 型ベクトル・プロセッサ・プロトタイプ, 並列処理シンポジウム JSPP '92 論文集, pp. 359-366 (1992).
15) Jouppi, N. P., Bertoni, J. and Wall, D. W.: A Unified Vector/Scalar Floating-Point Architecture, *Proc. 3rd Int'l Conf. on Architectural Support for Programming Languages and Operating Systems* (*ASPLOS-III*), pp. 134-143 (1989).
16) Nakamura, H., Imori, H., Nakazawa, K., Boku, T., Nakata, I., Yamashita, Y., Wada, H. and Inagami, Y.: A Scalar Architecture for Pseudo Vector Processing Based on Slide-Windowed Registers, *Proc. Int'l Conf. on Supercomputing*, pp. 298-307 (1993).

(平成5年3月30日受付)
(平成5年9月 8 日採録)



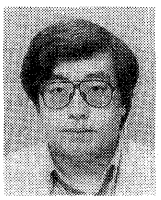
位守 弘充（正会員）

1969年生．1991年筑波大学情報学類卒業．1993年同大学院工学研究科修士課程修了．同年(株)日立製作所入社．計算機アーキテクチャ，並列処理の研究に従事．情報処理学会第44回全国大会奨励賞受賞．



中村 宏（正会員）

昭和38年生．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，平成3年同講師．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．電子情報通信学会，IEEE 各会員．



朴 泰祐（正会員）

1986年慶應義塾大学大学院工学研究科修士課程修了．1990年同大学院理工学研究科後期博士課程修了．工学博士．慶應義塾大学理工学部物理学科助手を経て現在，筑波大学電子・情報工学系講師．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．

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(株)日立製作所入社．TAC，HITAC 5020，E/F，8800/8700，M-200 H/280 H，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の研究開発に従事．工学博士．電子情報通信学会，IEEE，ACM 各会員．