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

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てベクトル計算を高速に処理する新しい擬似ベクトルプロセッサを提案する．提案するプロセッサは，スーパスカラ方式を前提としているが基本的にスカラアーキテクチャであり，ベクトル命令やベクトルレジスタを有するものではない．スカラプロセッサではキャッシュミス時の主記憶アクセスペナルティによる実効性能の低下が問題となる．ここで提案するプロセッサでは，データキャッシュの代りにレジスタウィンドウ方式により拡張した浮動小数点レジスタを採用し，さらに主記憶アクセスをパイプライン化することでこれを解決する．1つのベクトル命令の処理内容は複数のスカラ命令を垂直マイクロプログラム的に使用することにより擬似的に処理される．これらの特徴により，提案するプロセッサは既存のスカラアーキテクチャとの上位互換性を保つことが可能である．本論文では，提案するプロセッサのアーキテクチャとその処理原理を説明し，ベンチマークを用いた性能評価結果を示す．評価した結果，提案するプロセッサは主記憶アクセスペナルティが 20 CPU Cycle の時に，拡張を行わないスカラプロセッサに対して約 10 倍の性能，キャッシュへのプリフェッチを行うプロセッサに対しても約 1.4 倍の性能を達成することがわかった．また，30 CPU Cycle 程度までの主記憶アクセスペナルティをほぼ完全に隠せることがわかった．また，レジスタウィンドウの構成方法の相違による性能への影響についても検討した．これらの評価結果より，提案するプロセッサは主記憶アクセスペナルティによって実効性能が低下することなく，高速にベクトル計算を処理できると結論できた．

Evaluation of Pseudo Vector Processor Based on Register Window

Hiroshi Nakamura,† Hiromitsu Imori†† and Kisaburo Nakazawa†

This paper describes a new scalar architecture for high-speed pseudo vector processing based on register window and superscalar pipeline. Without using cache memory, the proposed architecture can tolerate the penalty of memory access latency by introducing register windows and pipelined memory. A single vector operation is realized by multiple scalar instructions as if they are vertical micro-instructions for the vector operation. The proposed architecture holds upward compatibility with existing scalar architectures. In this paper, the architecture, the principle, and the performance evaluations of the proposed processor are described. When the memory access latency is 20 CPU cycles, the performance of the proposed processor is about 10 times higher than an ordinary scalar processor and about 1.4 times higher than a hypothetical model extended with cache prefetching technique. It is also shown that the proposed processor can tolerate the memory access latency of about 30 CPU cycles. The effect of altering register window structure is also presented.

1. はじめに

近年，スカラプロセッサの性能は，集積回路技術の進歩およびスーパスカラ方式等の処理方式上の工夫により，著しく向上している．しかしメモリ素子のアクセスタイムはさほど改善されていないため，主記憶アクセスペナルティがもたらすプロセッサの実効性能への影響は相対的に増大している．この傾向は今後も続く[3]と考えられるため，プロセッサにデータや命令を効率良く供給するための方式上の工夫がより重要になっている．

現在のところほとんどすべてのスカラプロセッサは，この問題に対してキャッシュを用いることで対処している．しかし大規模な科学技術計算においては，多くの場合データキャッシュが有効に働かないことが報告されている[2),14)]．これは，計算のためのデータ領域がキャッシュ容量よりはるかに大きく，またデータ参照に局所性（特に，再利用されるという時間的局所性）が少ないためである．従ってこれらのアプリケー

† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba
†† 筑波大学大学院工学研究科
Doctoral Program of Engineering, University of Tsukuba

ションにおいてはキャッシュのミス率が高く，キャッシュを頼りにするプロセッサの性能は大きく低下する．科学技術計算用のベクトル方式スーパコンピュータではキャッシュを頼りにしていないことが，これを端的に示している．

この問題に対する解決法として，プリフェッチ機能をキャッシュに持たせ，必要なデータをあらかじめキャッシュに取り入れることにより主記憶アクセスペナルティを低減する手法が提案されている[7),9),15)]．しかしこの手法には以下の深刻な問題が伴う．第1の問題点は，非連続アドレスのアクセス時に不要なデータもプリフェッチする可能性である．その場合，不要なデータトラフィックが主記憶とキャッシュの両方に生じ，それらに要求されるスループットは厳しくなる．第2の問題点は，時間的局所性が無いために生じるキャッシュトラフィックの増大である．データに再利用性が無い場合，キャッシュは単なるデータの通り道にすぎない．その場合，主記憶／キャッシュ間の read/write とキャッシュ／プロセッサ間の read/write を考えると，最悪の場合キャッシュに要求されるスループットが主記憶のスループットの2倍になってしまう．キャッシュのスループットを高くするためには，例えば文献 6)，9) に述べられているように高価な multi-port cache あるいは multi-bank cache を用意する必要が生じる．第3の問題点は，データをプリフェッチする際に，有効なデータを追い出してしまう可能性である．特にキャッシュがダイレクトマップ方式である場合は line conflict によりこの追い出しが起こる可能性が高い．

我々が提案する擬似ベクトルプロセッサ[11),12)]は，データキャッシュの代りにレジスタウィンドウ方式により拡張した浮動小数点レジスタを採用することにより，スカラプロセッサで問題となるキャッシュミス時の主記憶アクセスペナルティによる実行性能の低下を防ぎ，大規模な科学技術計算を高速に処理できるスカラプロセッサである．

2. 擬似ベクトル処理方式

2.1 擬似ベクトル処理

ベクトルプロセッサとスカラプロセッサの相違点は，以下のようなデータ供給手法／能力に集約される．すなわち，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，主記憶とのデータ転送がベクトル命令によりパイプライン的に処理される．多数のベクトルレジスタを用意することによりベクトルロード／ストア命令のベクトル長を長くでき，さらに，chaining の機構により演算の実行と並行して行われる一種のベクトルレジスタへのプリロード機能が大きな役割を果たすため，主記憶へのアクセスのレーテンシが大きくても性能にあまり影響を与えない．一方，スカラプロセッサでは，内部での命令・演算の処理はかなり高度にパイプライン化され，特に最近はスーパスカラパイプライン方式が採用されつつあるにもかかわらず，一般に主記憶との間のデータ転送がパイプライン化されていない．そのため演算のためのデータ供給がボトルネックとなってしまう．これを補うものとしてキャッシュがあるが，これには冒頭に述べた問題点がある．

従ってスカラプロセッサにおいても，ベクトルプロセッサとの相違点である．

(1) キャッシュに代るものとして，ベクトルレジスタの役割を担うことのできる適当に多数のレジスタを用意
(2) 主記憶へのアクセスをパイプライン化
(3) レジスタへのプリロード機能の強化

を行えば，スーパスカラ方式の命令パイプライン構成のプロセッサでスカラ命令をベクトル命令の垂直マイクロプログラム的に用いることにより，1つのベクトル命令の処理内容を複数のスカラ命令で擬似的に実行できる（擬似ベクトル処理）と考えられる．

提案する擬似ベクトルプロセッサでは，この3点を既存の命令セットアーキテクチャとの上位互換性を保ちながら以下のように実現する．

(a) 命令フォーマット中のレジスタ指定フィールドが限られているため，命令セットアーキテクチャを変更せずにレジスタ数を増加することは一般に困難であるが，浮動小数点レジスタを複数のセット（レジスタウィンドウ）に分割することによりこの問題を解決する
(b) 通常のベクトルプロセッサと同様に，多重バンク構成をメモリモジュールをインタリーブさせ，メモリを擬似的にパイプライン化させる
(c) レジスタへのプリロード機能を実現する数種類の命令を追加する

2.2 Phase Pipelining

この3点の改良を加えて擬似ベクトルプロセッサで，科学技術計算を高速に処理する原理を説明する．

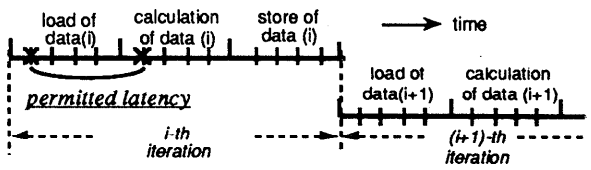
科学技術計算において最も計算能力を必要とする部

分はループ計算であり，各ループでの実行はデータのロード／計算／データのストアの3つの phase に概念的に分けられる．メモリがパイプライン化されメモリへのアクセス要求を連続時に行える場合，ループ実行は概念的に**図1**(a)のように表せる．この図において，例えば“load of data(i)”の部分は i 番目の繰り返しで用いられるデータに対するロード命令の発行を表す．要求されたデータが実際に計算に利用されるまでの時間間隔（図で×と×の間の時間間隔：以降 permitted latency と呼ぶ）が主記憶アクセスレーテンシより短い場合，計算を行いたい時点で必要なデータが主記憶から到着していないことになり，プロセッサの実効性能が低下することになる．この permitted latency はループアンローリングの手法を用いて各 phase を長くすることによりある程度長くできる．

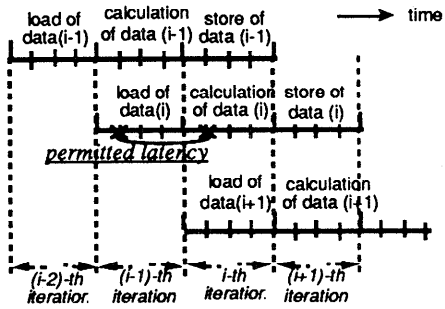
図1(a)の処理方式に対しsoftware pipelining の手法を用いれば，図1(b)のようにロード／計算／ストアの phase の実行を重ねることができる．明らかに(b)の方が(a)よりピーク性能は良いが，permitted latency は逆に(b)の方が短くなる．これは，ロード／計算／ストアの3つの phase が同一のレジスタ空間を用いるため各 phase が自由に使用できるレジスタが少なく，ループアンローリングの回数が減少して各 phase の長さが短くなってしまうからである．従って(b)の場合にも，主記憶アクセスレーテンシによる性能低下の影響が大きく現れてしまうことがある．

この問題を解決するため，1つのループ内において各 phase が命令で指定できる最大限のレジスタを活用できるようにした，図1(c)に示す phase pipelining の手法を我々は提案する．phase pipelining においては，レジスタは複数のセット（ウィンドウ）にわかれており，ある時点においてはただ1つのウィンドウのみが active である．この図では i 番目の繰り返しを実行している時にウィンドウ j が active であるが，この繰り返しにおいては，

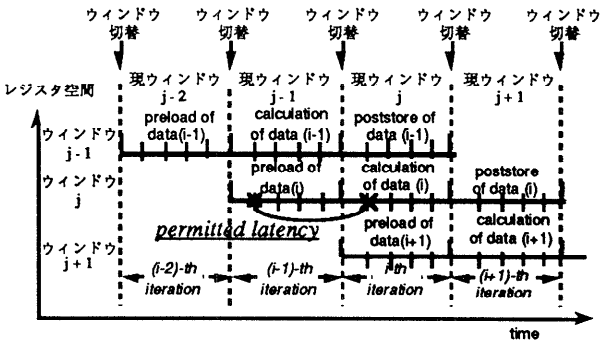
- 次の $(i+1)$ の繰り返しで用いるデータを次の $(j+1)$ ウィンドウへロード
- 今 active であるウィンドウ j を用いた計算



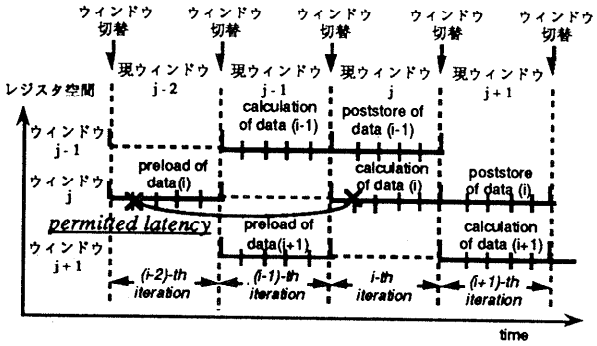
(a) 通常の実行
(a) Ordinary execution



(b) software pipelining による実行
(b) Execution in software pipelining



(c) 擬似ベクトル処理における phase pipelining
(c) Phase pipelining in pseudo vector processing



(d) 擬似ベクトル処理における拡張 phase pipelining
(d) Extended phase piplelining in pseudo vector processing

図1 擬似ベクトル処理の原理
Fig. 1 Principle of pseudo vector processing.

実行

- 前の $(i-1)$ の繰り返しで計算されたデータを前の $(j-1)$ ウィンドウからストア

の3つの phase が実行される．図1(b)の software pipelining との違いは，各 phase が異なるレジスタ空間を用いているためループアンローリングの手法が有効に活かされ，permitted latency を図1(a)と同等に長くできる点である．

また，2つ先のウィンドウへのロードを行うことにすれば，図1(d)に示す拡張 phase pipelining を実現することが可能である．この手法では permitted latency をさらに長くすることができ，主記憶アクセスレーテンシがさらに大きな場合にも効率よく主記憶アクセスレーテンシを隠すことができる．

3. 擬似ベクトル処理プロセッサのアーキテクチャ

提案する擬似ベクトルプロセッサのアーキテクチャは既存のスカラプロセッサのアーキテクチャを拡張することにより得られ，拡張前のスカラアーキテクチャに対して上位互換性を保てる．ここでは，拡張されるスカラアーキテクチャの例として Hewlett-Packard 社の PA-RISC 1.1 Architecture[4] を取り上げて説明する．他のスカラアーキテクチャに対しても同様の拡張を行うことにより擬似ベクトルプロセッサのアーキテクチャを得ることができる．

3.1 拡張するスカラアーキテクチャの概要

PA-RISC 1.1 Architecture はロード／ストアアーキテクチャであり，メモリと浮動小数点レジスタ間はロード／ストア命令でのみ4または8バイトのデータ転送ができる．浮動小数点レジスタは32個あるが，そのうち register 0-3 の4個は予約されておりユーザプログラムは残りの 28 個のみを用いる．浮動小数点演算命令として，加減算と乗算の両方を行う複合命令が用意されており，例えば “FMPYADD rm 1, rm 2, tm, ra, ta” という命令は ra ≠ tm, ta ≠ rm 1, rm 2, tm という条件の元で “FPR [tm] <- FPR [m 1] * FPR [m 2]; FPR [ta] <- FPR [ta] + FPR [ra]” の計算を行う．

3.2 擬似ベクトルプロセッサ実現のためのアーキテクチャ拡張

アーキテクチャ上の変更は，浮動小数点レジスタのウィンドウ化と，命令の付加のみである．

(1) 浮動小数点レジスタのウィンドウ化

浮動小数点レジスタは，元の PA-RISC 1.1 Architecture で指定できる 32 個のレジスタより多い数のレジスタを物理的に用意し，これらをオブジェクトコードから指定するために論理的にウィンドウ構造のものとして考える．また，現在アクティブなウィンドウ（現ウィンドウ）を指すポインタ CFRWP (Current Floating Register Window Pointer) を PSW (Program Status Word) 内に設ける．実際の命令実行時の物理レジスタは，命令で指定される論理的なレジスタ番号と CFRWP の値のペアで一意に指定される．レジスタウィンドウの構成例を**図2**に示す．この場合ウィンドウ数が4なので 2bit の CFRWP が必要である．この構成においては例えば window #1 の register #30 は window #2 の register #10 と同じで物理レジスタ番号は 50 となる．各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必要がある．global, overlapping, local の

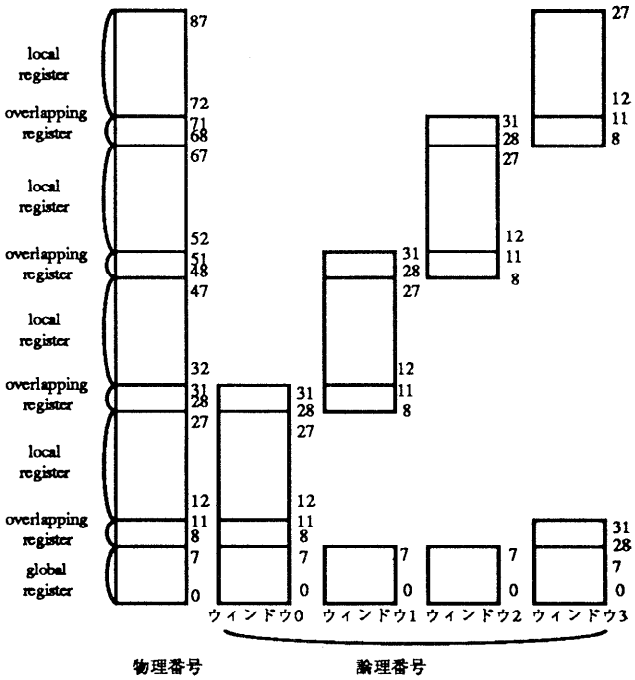


図2 レジスタウィンドウの構成
Fig. 2 Structure of register window.

3種類のレジスタの数やウィンドウ数等の構成については，他の構成もあり得るが，図1(d)に示した拡張 phase pipelining を行うためには最低4つのウィンドウが必要となる．これらのウインドウは，後述する命令によってポインタ（CFRWP）をインクリメントすることにより順次アクティブになるが，window #3 の次には window #0 がアクティブになるよう循環的に使用される．以降の説明では図2で示したウィンドウ構成を仮定するが，5章で示す性能評価では，ウィンドウ構成を変化させた場合も検討する．

（2） 追加命令

以下の命令を新たに追加する．

- CFRWPenable：特権命令であり，レジスタウィンドウが利用可能か否かを設定する．もしこの命令によりレジスタウィンドウが利用不可能になっている場合は，window #0 のレジスタのみを用いる．また，レジスタウィンドウが利用可能であっても以下に述べる追加命令を一度も使用しないオブジェクトコードではアクティブなウィンドウは window #0 から動くことはない．これらの場合，提案する新しいアーキテクチャはレジスタウィンドウを用いていないアーキテクチャと互換性を持つことになる．一方，この命令によりレジスタウィンドウを利用可能にする時は，初期値として CFRWP に0を設定する．
- CFRWPinc：非特権命令であり，CFRWP の値を modulo 4 で1インクリメントする．
- FRPreload：非特権命令．主記憶からのデータを，現在アクティブであるウィンドウの次のウィンドウ内の指定レジスタに転送する．例えば CFRWP＝0 の時に 'FRPreload r12, (mem)' を実行すると，データは次のウィンドウ（window #1）の register #12 すなわち物理番号 32 のレジスタに転送される．この命令では，キャッシュヒット時にはキャッシュにあるデータを用いるが，キャッシュミスの時にはキャッシュの replacement は行われず，この命令でレジスタにロードされるデータはキャッシュに書き込まれない．従ってその時はキャッシュ内のデータは全く変化しない．
- FRPreload-E：非特権命令．拡張 phase pipelining を実現するための命令．主記憶からのデータを，次の次のウィンドウ内の指定レジスタに転送する．例えば CFRWP＝0 の時に 'FRPreload-E r12, (mem)' を実行すると，データは次の次のウィンドウ（window #2）の register #12 すなわち物理番号 52 のレジスタに転送される．この命令も FRPreload 同様，キャッシュミスの時にはキャッシュの replacement は行われず，キャッシュ内のデータは全く変化しない．
- FRPoststore：非特権命令．現在アクティブであるウィンドウの1つ前のウィンドウ内の指定レジスタから主記憶にデータを転送する．キャッシュミスの時には FRPreload 命令と同様，キャッシュ内のデータは変化しない．

4. 擬似ベクトル処理例

レジスタウィンドウを用いた擬似ベクトル処理の原理を説明するため，図2のレジスタ構成における内積計算の例を**図3**に示す．説明を簡単にするため，ここでは分岐命令とインデックス計算（PA-RISC 1.1 ではアドレス修飾と同時に指定可能）は省略されており，1回しかループアンローリングを行っていない．図3は，左側が浮動小数点演算命令，右側が FRPreload 命令を示しており，これらの命令がスーパスカラ処理により並列に実行されることを表している．この処理では，あるループにおいて $a(i)$, $b(i)$, $a(i+1)$, $b(i+1)$, に対する計算と，$a(i+2)$, $b(i+2)$, $a(i+3)$, $b(i+3)$ の次ウィンドウへのプリロードが行われる．また，$a(i)*b(i)$ と $a(i+1)*b(i+1)$ の和を取る計算は，software pipelining の手法によって実際には次のループで行われる．

図4はこの処理におけるレジスタ割当を示している．この図からわかるように，overlapping register により software pipelining 的な計算を行うことができ，また global register により，c0, c1 といったすべてのループに共通なデータを表すことができる．このコードでは，あるデータを用いる演算命令はそのデータに対する FRPreload 命令より 3CPU Cycle 以上遅れて発行されるので permitted latency は3 CPU Cycle である．実際には，このコードでは未使用

	浮動小数点演算		FRPreload
	mult	add	
1		c0=c0+a(i-2)*b(i-2)	FRPreload r24, a(i+2)
2	a(i)*b(i)	c1=c0+a(i-1)*b(i-1)	FRPreload r25, b(i+2)
3	a(i+1)*b(i+1)		FRPreload r26, a(i+3)
4	CFRWPinc		FRPreload r27, b(i+3)

図3 内積計算のコード例
Fig. 3 Code example for inner product calculation.

のレジスタがまだたくさん残されており，ループアンローリングの手法をさらに用いることにより各ループの実行サイクルを長くできるため，permitted latency はさらに長くなる．

5. 評　価

5.1 評価対象モデル

提案する方式の有効性を検証するため，以下の5種類のプロセッサモデルに対して評価を行う．**表1**にこれらのモデルの特徴をまとめて示す．

〈比較1〉 オリジナルの PA-RISC 1.1 Architecture で，メモリはパイプライン化されていない．

〈比較2〉 〈比較1〉に対し，メモリのパイプライン化と Prefetch 命令の導入による拡張を行い，データキャッシュへのプリフェッチをソフトウェア的に可能にしたもの．

〈提案1〉 本稿で提案する擬似ベクトルプロセッサアーキテクチャ．〈比較1〉に対し，メモリのパイプライン化および浮動小数点レジスタのウィンドウ化の拡張を行い，図1(c)の phase pipelining 処理を実現したもの．レジスタウィンドウ構成は図2のものを採用している．FRPreload-E 命令による拡張 phase pipelining は実現されていない．

〈提案2〉 本稿で提案する擬似ベクトルプロセッサアーキテクチャ．〈提案1〉に対し，FRPreload-E 命令を導入し，図1(d)に示した拡張 phase pipelining を実現したもの．レジスタウィンドウ構成は図2のものを採用している．

〈理想〉 〈比較1〉において，必要な全データがあらか

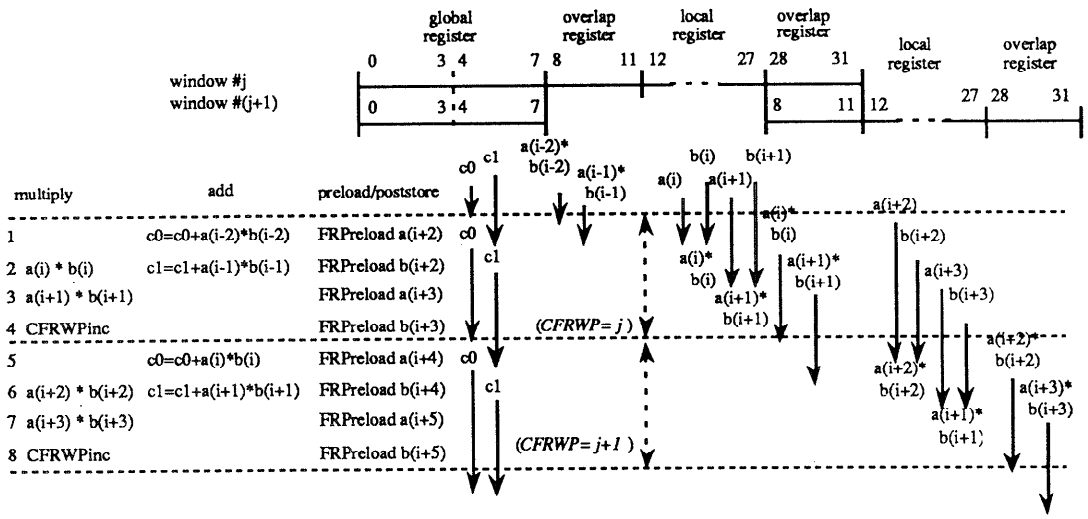


図 4 内積計算におけるレジスタ割当
Fig. 4 Register allocation for inner product calculation.

表 1 評価対象プロセッサモデルの特徴
Table 1 Summary of evaluated processor model.

プロセッサモデル	比較 1 (original)	比較 2 (cache prefetch)	提案 1 (register window)	提案 2 (register window)	理想 (always cache hit)
アーキテクチャ	PA-RISC 1.1	PA-RISC 1.1 +prefetch 命令	拡張 PA-RISC 1.1	拡張 PA-RISC 1.1	PA-RISC 1.1
レジスタ	変更無し	変更無し	window 化	window 化	変更無し
主記憶	非パイプライン化	パイプライン化	パイプライン化	パイプライン化	access 無し
主記憶への特別なアクセス命令	無し	キャッシュへの prefetch 命令	preload および poststore 命令	preload, poststore, および拡張 preload 命令	無し
データキャッシュ	conventional	multi-ported fully associative	conventional	conventional	conventional (no cache miss)

じめキャッシュに存在していることを仮定した理想的なもの.

5.2 評価ベンチマーク

Livermore Loop Kernel, Linpack benchmark, および QCD 計算の主要部から取り出した，**図5**に示す5つの典型的な応用例に対して評価を行った．ただし，データ領域の大きさを実際のアプリケーションに近付けるために，繰り返しの回数は増加させている．従って，ループ部の処理に入る前のプロローグ部とループから抜けた後のエピローグ部の処理の影響は無視している．

5.3 評価環境

評価は，オブジェクトコードを命令パイプラインのステージレベルでシミュレーションすることによって行った．オブジェクトコードはハンドコンパイルによって生成したが，評価を公平にするため各プロセッサモデルに対して最適なオブジェクトコードを各々作成している．従って，ループアンローリングは各プロセッサモデルにおいて可能な限りのレジスタを使用し，最大限行われている．また，〈比較2〉においてはデータのアクセスパターンを完全に予測できるとし，prefetch 命令の挿入は最小限にとどめてある．評価において以下のことを仮定した．特に明記していない場合は，すべてのプロセッサモデルに対する仮定を表す．

- 命令発行：2命令同時発行のスーパスカラ方式である．stall が発生すると後続命令はインタロックされる (in-order)．命令はロード／ストア命令 (Prefetch, FRPreload, FRPreload-E, FRPoststore を含む)，浮動小数点演算，その他 (整数演算，branch,

```
(a) Livermore Kernel #3
      do 1  k = 1, n
            Q = Q + Z(k) * X(k)
(b) Livermore Kernel #4
      do 2  k = 5, n, 5
            T = T - X(lw) * Y(k)
            lw = lw + 1
(c) Livermore Kernel #7
      do 3 I = 1, n
            X(I) = U(I) + R* Z(I) + R*Y(I) ) +
            T*( U(I+3) + R*( U(I+2) + R*U(I+1) ) +
            T*( U(I+6) + R*( U(I+5) + R*U(I+4) ) ) )
(d) DAXPY (Linpack Benchmark)
      do 4 I = 1, n
            Y(I) = A * X(I) + Y(I)
(e) QCD (Quantum Chromo Dynamics)
      do 5 I = 1, n
            H(I) = H(I) + S*( U1(I)*G1(I) +U2(I)*G2(I) )
```

ただし，(a)-(d)の各データと(e)のSは倍精度の実数，(e)のその他のデータは倍精度の複素数

図5 評価ベンチマーク
Fig. 5 Target benchmark.

CFRWPinc 等) の3つのカテゴリーに大別され，同時に発行される命令は異なるカテゴリーに属さなければならない．なお，3.1 節で述べたとおり浮動小数点演算の加減算と乗算は合わせて1つの複合命令として実行される．

- データキャッシュ：〈提案1〉〈提案2〉以外のモデルにおいて，以下のことを仮定する．ブロックサイズは 16 B である．また line conflict は起こらない．これは fully associative cache を仮定することと等価である．〈比較1〉においては cold cache を想定する．また〈比較2〉においては multi-port cache を仮定し，キャッシュのスループットは十分確保されているとする．
- 主記憶：〈比較2〉〈提案1〉〈提案2〉では，主記憶はパイプライン化されており，prefetch, preload 命令は毎サイクル発行できるとする．パイプラインメモリは多重バンク構成により擬似的に実現することを想定する．しかし，ここではバンクコンフリクトが発生しないよう各データが最適にメモリ上に割り当てられていることを仮定し，バンクコンフリクトの影響は考慮しない．
- データ依存関係：データ依存関係のある2命令間では，先行命令が浮動小数点演算なら 5 CPU Cycle 以上，先行命令が Load, Prefetch, FRPreload, FRPreload-E ならば cache hit 時には 2CPU Cycle 以上，cache miss 時には memory access latency に等しい CPU Cycle 以上，後続命令の発行を先行命令より遅らせる．ただし，〈比較2〉のモデルにおいては，データ依存関係による待ちが生じないよう，prefetch 命令の十分な先行発行をコンパイラがスケジューリングできると仮定する．
- 制御依存関係：分岐命令のペナルティは，delayed branch により回避できると仮定する．
- 命令キャッシュ：必要な命令はすべてあらかじめ命令キャッシュ内にあると仮定する．

5.4 評価結果

(1) 各ベンチマークに対する性能

周波数 100 MHz 主記憶アクセスレーテンシ 20CPU Cycle (200 nsec) を仮定した時の，各プロセッサモデルの各ベンチマークに対する MFLOPS 値とその調和平均を**図6**に示す．通常のキャッシュを用いた〈比較1〉のモデルでは，キャッシュミス時のペナルティにより大きく性能が低下し，調和平均で比べると〈理想〉モデルのわずか 10% である．〈比較2〉の性能が

〈理想〉モデルに対し調和平均で30%低下する原因は，キャッシュへの prefetch 命令の挿入による実行サイクルの増大である．一方，〈提案1〉および〈提案2〉の性能は調和平均で各々〈理想〉モデルの 91%，99% であり，〈比較1〉に対して 9.4 倍，10.2 倍，〈比較2〉に対して 1.3 倍，1.4 倍である．〈提案2〉のわずか1%の性能低下は，レジスタウィンドウ構造が固定であるため，命令のスケジューリングに自由度が無いために実行サイクルが長くなるためである．〈提案1〉の性能低下は，Livermore Loop #3，#4 において permitted latency が 20 CPU Cycle 未満であるため，完全には主記憶アクセスレーテンシを隠せなくなるためである．

（2） 主記憶アクセスレーテンシを変化させた場合の性能

図7は，主記憶アクセスレーテンシを変化させた時の各モデルの性能を，〈理想〉モデルに対する相対性能値で示している．相対性能は，5つのベンチマークにおける性能の調和平均より求めた．〈理想〉モデルでは all cache hit が仮定されているため，主記憶アクセスレーテンシを変化させても性能は一定である．この図からわかるように，拡張を施さない〈比較1〉のモデルでは主記憶アクセスレーテンシの増加に対し性能が著しく低下する．〈提案1〉〈提案2〉においては，アクセスレーテンシが大きい時に preload したデータが演算開始時に到着していない場合，性能が低下する．しかし図7よりわかるように，拡張 phase pipelining を採用する〈提案2〉のモデルでは，レーテンシが 30 CPU Cycle までは全く性能は低下しない．〈比較2〉の性能がレーテンシの値によらず一定である理由は，キャッシュへの prefetch 命令を副作用無しに十分前もって発行できるという楽観的な仮定をしているためである．しかしそれでも，図7よりわかるように 50 CPU Cycle 以下のレーテンシに対しては，〈提案2〉は楽観的な仮定をした〈比較2〉よりも性能が良い．

（3） ウィンドウ構成を変化させた時の擬似ベクトルプロセッサの性能

ここまでは擬似ベクトルプロセッサのレジスタウィンドウの構成は図2のとおりで固定であった．ここでは，さらにウィンドウ構成を変化させた場合の性能を示す．**図8**は，〈提案1〉モデルにおいて主記憶アクセスレーテンシ 20 CPU Cycle を仮定した時の，ウィンドウ構成を変えた場合の各ベンチマークに対する

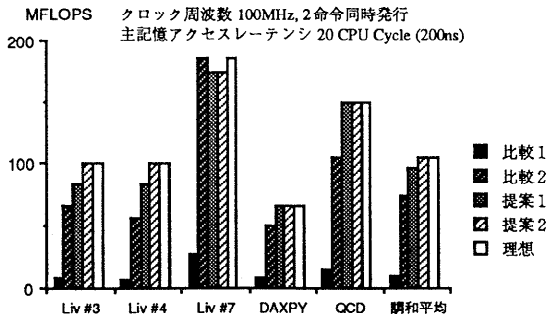


図 6 各プロセッサモデルの性能
Fig. 6 Performance of each processor model.

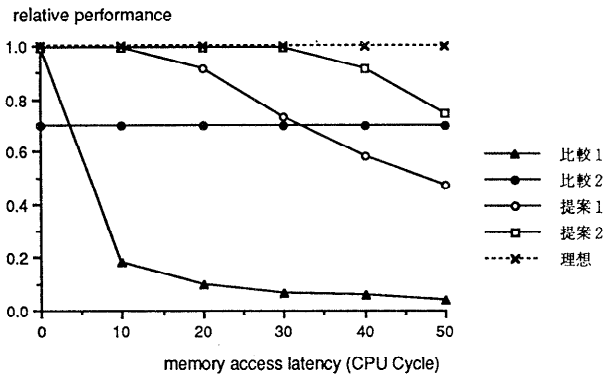


図 7 主記憶アクセスレーテンシを変化させた時の各モデルの相対性能
Fig. 7 Relative performance of each processor model under variable memory access latency.

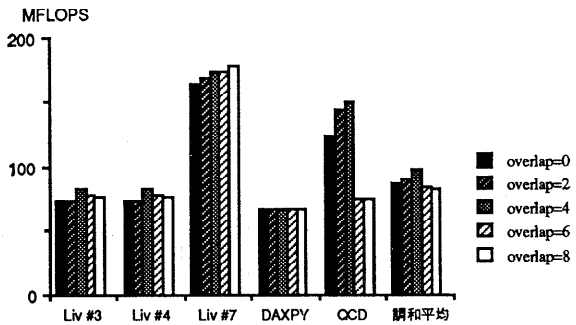


図 8 ウィンドウ構成を変化させた時の〈提案1〉の性能（主記憶アクセスレーテンシ 20 CPU Cycle）
Fig. 8 Performance of 〈Proposed 1〉 with altering window structure. (memory access latency=20 CPU cycle)

MFLOPS 値とその調和平均を示す．図 9, 10 は，同様に〈提案 2〉モデルにおいてウィンドウ構成を変えた時，主記憶アクセスレーテンシを各々 20 CPU Cycle, 50 CPU Cycle と仮定した場合の性能を示す．さらに，主記憶アクセスレーテンシを変化させた時の〈提案 2〉モデルの性能（5 つのベンチマークに対する性能の調和平均）を，ウィンドウ構成をパラメータとして評価した結果を図 11 に示す．ウィンドウ構成の変更は overlapping register 数を増減することで実現した．ウィンドウの総数 4, global register の数 8 は不変である．図 8〜11 における "overlap=n" の n は，1 つ先のウィンドウと共有する overlapping register の数が n 個であることを表す．例えば図 2 のウィンドウ構成は "overlap=4" の場合になる．いずれの場合もウィンドウ数が一定なので，n の値が増加すれば物理レジスタの総数は減少することになる．図 11 よりわかるように，overlapping register が多いウィンドウ構成においては，主記憶アクセスレーテンシの増大による性能低下が顕著になる．これは，overlapping register が多いと各ウィンドウが独立に利用できるレジスタ数が減少し，図 1 における phase pipelining においてウィンドウ間の干渉が生じるためにループアンローリング回数が減り，結果的に permitted latency が短くなってしまうためである．図 10 よりわかるように，〈提案 2〉モデルにおいてレーテンシが 50 CPU Cycle の時では，overlapping register の数が少ないほど性能は高いという傾向がほぼ成り立つ．一方，レーテンシが小さい場合は，図 8 および図 9 よりわかるように，望ましいウィンドウ構成は実行対象となる計算の性質に依存する．図 11 よりわかるように，評価対象の 5 つのベンチマークに対する〈提案 2〉モデルの調和平均は，主記憶アクセスレーテンシが 0〜30 CPU Cycle の時では図 2 で最初に仮定したウィンドウ構成を採用する場合が最も性能が良かった．従って，これらのベンチマークに対しては，当初仮定したウィンドウ構成は妥当なものと結論できる．

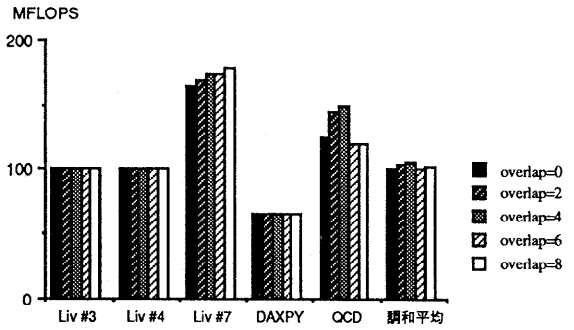


図 9　ウィンドウ構成を変化させた時の〈提案 2〉の性能
（主記憶アクセスレーテンシ 20 CPU Cycle）
Fig. 9　Performance of 〈Proposed 2〉 with altering window structure. (memory access latency=20 CPU cycle)

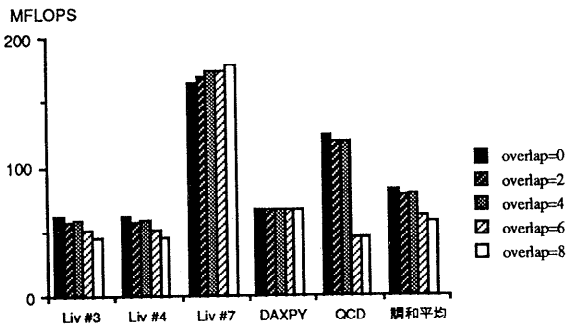


図 10　ウィンドウ構成を変化させた時の〈提案 2〉の性能
（主記憶アクセスレーテンシ 50 CPU Cycle）
Fig. 10　Performance of 〈Proposed 2〉 with altering window structure. (memory access latency=50 CPU cycle)

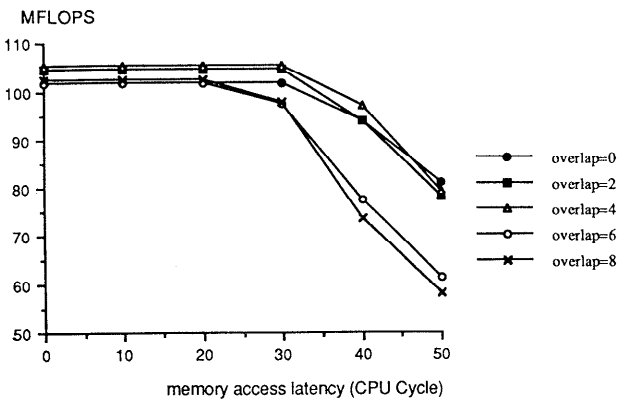


図 11　ウィンドウ構成と主記憶アクセスレーテンシを
変化させた時の〈提案 2〉の性能
Fig. 11　Performance of 〈Proposed 2〉 with altering window structure under variable memory access latency.

6. 考　　察

6.1 擬似ベクトル処理の有効性

（1） キャッシュプリフェッチとの比較

主記憶アクセスレーテンシが 20 CPU Cycle の時，キャッシュへのプリフェッチを行う〈比較2〉に比べて我々の提案する〈提案1〉〈提案2〉のモデルは各々1.3 倍，1.4 倍の性能を達成する．これは，我々が提案する擬似ベクトルプロセッサでは Prefetch 命令の挿入による命令ステップ数の増加がないためである．キャッシュへのプリフェッチにはここで用いたソフトウェアによるもの以外にハードウェアによるものも提案されている[1),7)]．ハードウェアによるプリフェッチを用いた場合，命令ステップの増加は無くなるものの，今度はデータのアクセスパターンを完全には予測できないという欠点がある．

さらに，仮にハードウェアにより命令ステップの増加無しに完全にデータのプリフェッチが行えると仮定しても，キャッシュプリフェッチの手法には1章で指摘したように以下の問題点がある．

- キャッシュにおけるデータトラフィックの増加
- 不要なデータのプリフェッチ
- 必要なデータの追い出し

提案する方式は，データを主記憶からキャッシュを介さず直接レジスタへ転送することによりこれらの問題点をすべて解決したものである．

（2） ベクトル計算機との比較

ベクトル計算機と比較しても以下の長所がある．

- ベクトルレジスタを用いていないので strip-mining の手法が不要である．
- ベクトル計算機で仮想メモリをサポートする場合は，複数のデータが1命令で転送されるために page fault 等の割込発生時の処理が難しくなるが，本方式では通常のスカラプロセッサと同様に仮想メモリをサポートできる．
- ベクトル命令のための制御レジスタの初期設定が不要なので処理の立上り時のオーバヘッドが小さい．従って半性能ベクトル長が短くなる．

（3） レジスタウィンドウ構成の妥当性

5章で述べたとおり，最適なレジスタウィンドウの構成は，実行する処理の性質と主記憶アクセスレーテンシに依存する．定性的に言えば，主記憶アクセスレーテンシの影響を隠すためには，物理レジスタの総数が多い方が permitted latency が長くなるので望ましい一方で，主記憶アクセスレーテンシが permitted latency 以下の場合には，必要な overlapping register の数は実行する処理に依存する．従って，ウィンドウ構成を可変にするようなアーキテクチャの拡張も考慮に値するであろう．

6.2 実現の難易性

追加すべきハードウェアは数十のレジスタ，論理レジスタ番号から物理レジスタ番号を得るための変換回路，および物理レジスタ内のデータ依存関係を処理するための制御論理の拡張等であり，これらを実現するための chip 上の面積はいずれも比較的小さく chip 上の実装は容易でクロック周波数にも影響は与えない．もっともコストのかかるのはパイプラインメモリであるが，これはメモリスループットの向上のために本質的に必要不可欠と考えられる．

6.3 他の研究との比較

ここで提案した phase pipelining による擬似ベクトル処理方式は，元々一般の命令のレジスタ指定フィールドが十分大きなビット数を持ち，命令で直接指定できるレジスタ数が十分大きい場合（例えば 88 レジスタ）での software pipelining/unrolling 手法による最適化にも適用されるものである．しかし，本論文の特徴は命令のレジスタフィールドのビット数が十分大きくない既存のアーキテクチャに対し，レジスタウィンドウ方式で物理レジスタ数を拡張し，既存のアーキテクチャと上位互換性を保ちながら擬似ベクトル処理を可能としたところにある．

レジスタウィンドウの考え方は既に RISC-I[13)] や SPARC プロセッサ[16)]で採用されており新しいものではない．しかし，これらのプロセッサではウィンドウを procedure call/return の高速化に用いており，本提案の目的とは全く異なる．

ベクトル処理とスカラ処理を統一的に扱うアーキテクチャは MultiTitan[8)] で採用されており，また文献10) でもハイパスカラプロセッサとして提案されている．Multititan は Preload/Poststore に類似した概念を持っており，目的は我々と似ている．しかしベクトル命令とベクトルレジスタを持つ点が我々の提案とは異なる．ハイパスカラプロセッサの場合，ベクトル命令の処理をスカラ命令の組合せで擬似的に行うことを目指している点は類似しているが，主記憶とのデータ転送はキャッシュを介して行われており，主記憶アクセスのペナルティの軽減については考慮されていない．

キャッシュを用いないで，データをパイプライン的にレジスタへ直接挿入するという方法は i 860 プロセッサ[5]でも採用されている．しかし，i 860 においては，ロードパイプラインに必要とされるパイプラインの段数（マシンサイクル数）が，オブジェクトコードに陽に表れる必要がある．従って，パイプライン段数を変化させた場合，以前のオブジェクトは正しい結果を与えなくなってしまう．提案する擬似ベクトル処理においては，このような問題は存在しない．

7. まとめ

浮動小数点レジスタのウィンドウ化とメモリのパイプライン化により，既存のアーキテクチャと上位互換性を保ちながらベクトル処理を高速に処理する新しいアーキテクチャを提案し，ベンチマークを用いた性能評価を行った．評価結果より，提案するアーキテクチャは，主記憶アクセスペナルティが 20 CPU Cycle の時，拡張前のスカラプロセッサに対し約 10 倍の性能，キャッシュへのプリフェッチを行うプロセッサに対しても約 1.4 倍の性能を達成することがわかった．また，30 CPU Cycle 程度までの主記憶アクセスペナルティをほぼ完全に隠せることがわかった．これらの評価結果より，提案する擬似ベクトルプロセッサは大規模な科学技術計算に適していると結論できる．

謝辞 本研究に関し有益な御意見を頂いた，筑波大学中田育男教授，朴泰佑講師，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 CP-PACS グループおよび GNOH グループの方々に深謝いたします．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 04NP0601）による．

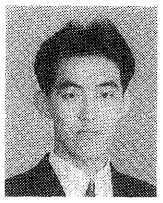
参考文献

1) Baer, J. L. and Chen, T. F.: An Effective On-Chip Preloading Scheme to Reduce Data Access Penalty, *Proc. of Supercomputing '91*, pp. 176-186 (1991).
2) Callahan, D. and Porterfield, A.: Data Cache Performance of Supercomputer Applications, *Proc. of Supercomputing '90*, pp. 564-572 (1990).
3) Hennessy, J. L. and Jouppi, N. P.: Computer Technology and Architecture: An Evolving Interaction, *IEEE Computer*, Vol. 24, No. 9, pp. 18-29 (1991).
4) Hewlett-Packard Company: PA-RISC 1.1 Architecture and Instruction Set Reference Manual, Manual Part Number 09740-90039 (1990).
5) Intel Corporation: i 860 64-Bit Microprocessor Programmer's Reference Manual, ISBN 1-55523-080-6 (1989).
6) 位守弘充，伊藤元久，中村 宏，中澤喜三郎：擬似ベクトル処理向きメモリアーキテクチャの一提案，情報処理学会研究会報告 ARC-91-8 (1991).
7) 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).
8) Jouppi, N. P., Bertoni, J. and Wall, D. W.: A Unified Vector/Scalar Floating-Point Architecture, *Proc. of ASPLOS-III*, pp. 134-143 (1989).
9) 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).
10) 村上和彰：ハイパースカラ・プロセッサ・アーキテクチャー命令レベル並列処理への第5のアプローチ—，並列処理シンポジウム JSPP '91 論文集，pp. 133-140 (1991).
11) 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 (1991).
12) Nakazawa, K., Nakamura, H., Imori, H. and Kawabe, S.: Pseudo Vector Processor Based on Register-Windowed Superscalar Pipeline, *Proc. of Supercomputing '92*, pp. 642-651 (1922).
13) Patterson, D. A. and Sequin, C. H.: RISC I: A Reduced Instruction Set VLSI Computer, *Proc. 8th Int'l Symp. on Computer Architecture*, pp. 443-457 (1981).
14) Simmons, M. L. and Wasserman, H. J.: Performance Evaluation of the IBM RISC System/6000: Comparison of an Optimized Scalar Processor with Two Vector Processors, *Proc. of Supercomputing '90*, pp. 132-141 (1990).
15) Smith, A. J.: Cache Memories, *Computing Surveys*, pp. 473-530, ACM (1982).
16) Sun Microsystems: The SPARC Architectural Manual, Version 8, Part No. 800-1399-09 (1989).

(平成4年 9 月16日受付)
(平成4年12月10日採録)

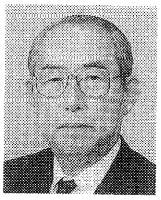
中村　宏（正会員）

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．IEEE 会員．



位守　弘充（正会員）

1969 年生．1991 年筑波大学第三学群情報学類卒業．現在同大学院工学研究科修士課程に在籍中．計算機アーキテクチャ，並列計算機の研究に従事．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-200 II/280 II, 680 H, 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．