計算機アーキテクチャ 94－3
（1992．6．12）

マイクロベクトルプロセッサ・アーキテクチャの検討

村上和彰　橋本隆　弘中哲夫　安浦寛人

九州大学 大学院総合理工学研究科

E-mail: {murakami, hashimot, hironaka, yasuura}@is.kyushu-u.ac.jp

マイクロベクトルプロセッサ構築に適したベクトル・アーキテクチャについて検討する．マイクロベクトルプロセッサは，I/Oピン数の制約によりチップ外メモリ・バンド巾を大幅に節約しなければならない．さらに，従来にも増して，ベクトル化可能範囲を拡大し，かつ，スカラ実行部分の性能を向上させる必要がある．本稿では，このようなマイクロベクトルプロセッサに有効なアーキテクチャ機能として，FIFOベクトル・レジスタ，ベクトル命令レベル・マルチスレッド処理，および，4種類の命令セット・アーキテクチャ（ISP_p, ISP_c, ISP_s, ISP_u）を提案し，その効果をシミュレーションにより評価している．

その結果，命令セット・アーキテクチャに関しては，ISP_u（unified scalar/vector ISP）モデルが最も性能が良く，ISP_s（separate scalar&vector ISP）モデル（従来のスーパーコンピュータのモデル）の1.03倍（相乗平均），ISP_p（purely scalar ISP）モデルの1.80倍（相乗平均）の性能であった．また，ISP_uモデル上で，ベクトル命令レベル・マルチスレッド処理により44%（相乗平均）の性能向上，FIFOベクトル・レジスタにより11%（相乗平均）の性能向上が得られた．

これより，マイクロベクトルプロセッサのアーキテクチャとして，ISP_uの**SIVR**（Scalar Instructions – Vector Registers）機能，および，ベクトル命令レベル・マルチスレッド処理機能が適しているとしている．

Micro-vectorprocessor Architectures

Kazuaki Murakami　Takashi Hashimoto　Tetsuo Hironaka　Hiroto Yasuura

Department of Information Systems
Interdisciplinary Graduate School of Engineering Sciences
Kyushu University
Kasuga-shi, Fukuoka 816 Japan

E-mail: {murakami, hashimot, hironaka, yasuura}@is.kyushu-u.ac.jp

This paper proposes and examines some architectural features suitable for vector microprocessors, which should save the off-chip memory bandwidth by exploiting the on-chip register bandwidth instead. Also, they should broaden the vectorizable part and improve the scalar performance further. Those features include FIFO vector registers, multithreading, and several ISP architectures.

First, the paper compares the effects of three ISP architectures: ISP_p (purely scalar ISP), ISP_s (separate scalar&vector ISP), and ISP_u (unified scalar/vector ISP). The ISP_u is the fastest model with being 1.80 times (geometric mean) faster than the ISP_p and 1.03 times (geometric mean) faster than the ISP_s. Next, for the ISP_u model, the paper investigates the effects of multithreading and FIFO vector registers. Multithreading and FIFO vector registers improve performance by 44% (geometric mean) and by 11% (geometric mean), respectively.

This paper concludes that the **SIVR**(Scalar Instructions – Vector Registers) capability of the ISP_u and the multithreading at the vector instruction level are good candidates for the architectural features suitable to vector microprocessors.

1 はじめに

ベクトル処理機能を1チップに集積したマイクロプロセッサをマイクロベクトルプロセッサ（*micro-vectorprocessor*）と呼ぶことにする．筆者らの知る限りでは，現在のところ以下の3つのマイクロベクトルプロセッサが存在する（表1参照）．

- NTT VP (Pipelined Vector Processor)[10]
- 日電 VPP (Vector Pipelined Processor)[11]
- 富士通 VPU (Vector Processing Unit) MB92831[8]

このようなマイクロベクトルプロセッサは，現在，RISC→スーパースカラ／スーパーパイプラインと高速化の一途を辿っている高性能汎用マイクロプロセッサ[4]の1つの最終到達目標と言えよう．そして，近い将来，エンジニアリング・ワークステーションの計算エンジンから大規模マルチプロセッサの要素プロセッサまで，広範な利用が予想される．

本稿では，このようなマイクロベクトルプロセッサを構築するのに適したベクトル・アーキテクチャを検討する．まず，2章および3章で，ベクトル・アーキテクチャ設計上の主要な検討項目であるオペランド・アドレッシングおよび命令セット・アーキテクチャについて検討を行い，種々の提案を行う．4章で，評価モデルを定義する．そして，5章で，シミュレーションによる評価を行い，その結果について考察する．

2 検討項目1

2.1 オペランド・アドレッシング

ベクトル・アーキテクチャは，ベクトル演算のオペランドであるベクトルデータをメモリおよびレジスタのいずれに置くかで，次の3方式に分類される．

- メモリ-メモリ演算方式（**SS**: *storage-storage architecture*）：CDC Star-100，CDC Cyber 200 Model 205，等で代表される方式で，2つのソース・オペランドと1つのデスティネーション・オペランドいずれもメモリ上に置いてベクトル演算を行う．
- レジスタ-レジスタ演算方式（**RR**: *register-register architecture*）：Cray-1で代表される方式で，2つのソース・オペランドと1つのデスティネーション・オペランドいずれもレジスタ（これをベクトル・レジスタと呼ぶ）上に置いてベクトル演算を行う．ベクトル演算の前後に，ベクトルデータのベクトル・レジスタへのロード，および，ベクトル・レジスタからのストアが必要で，「ロード／ストア・アーキテクチャ」とも呼ぶ．Cray以外に，富士通VP，日立S，日電SX，等の大部分のスーパーコンピュータが本方式を採用している．
- レジスタ-メモリ演算方式（**RS**: *register-storage architecture*）：上記2方式の折衷方式で，1つのソース・オペランドと1つのデスティネーション・オペランドはベクトル・レジスタ上に，もう1つのソース・オペランドはメモリ上に置いてベクトル演算を行う．IBM 3090VFが本方式を採用している．

レジスタ-レジスタ演算方式は，メモリ-メモリ演算方式に対して，以下の優位性を持つ．

a. 要求メモリ・バンド巾：1回の演算に対して，通常3個のオペランド（ソース・オペランド2個とデスティネーション・オペランド1個）が必要である．これは，メモリ-メモリ演算方式においては，1回のベクトル演算当り3回のメモリ・アクセス（2回のベクトル・ロードと1回のベクトル・ストア）を行うことを要求する．したがって，実効演算スループットは，$\frac{\text{メモリ・バンド巾}}{3}$で抑えられる．一方，レジスタ- レジスタ演算方式では，ベクトル演算に先だってベクトルデータをメモリからベクトル・レジスタにロードしておき，すべての演算はレジスタ-レジスタ間で行う．中間結果はベクトル・レジスタ上に置いておき，最終結果のみをメモリにストアする．また，1個のソース・オペランドを複数の演算で用いることも可能である．このように，レジスタ-レジスタ演算方式ではメモリ・トラフィックを減少させ，実効演算スループットを$\frac{\text{メモリ・バンド巾}}{3}$より大きくすることが可能となる．

上記のことを換言すれば，所望の実効演算スループット（ST : $SustainedThroughput$）を一定と仮定した場合に要求される要求メモリ・バンド巾（MB : $MemoryBandwidth$）は，メモリ-メモリ演算方式（**SS**）およびレジスタ-レジスタ演算方式（**RR**）とで，下式のようになる．

$$\begin{aligned} ST \times 3 &\leq MB_{SS} \\ ST &\leq MB_{RR} \leq ST \times 3 \end{aligned}$$

つまり，レジスタ-レジスタ演算方式は，メモリ-メモリ演算方式ほど高いメモリ・バンド巾を必要としない．

b. チェイニング機能：チェイニング（*chaining*）とは，フロー依存関係にある2つのベクトル命令（たとえば，ベクトル・ロード→ベクトル演算，ベクトル演算→ベクトル演算，ベクトル演算→ベクトル・ストア）をオーバラップ実行する機能である．一般に，メモリ-メモリ演算方式では暗黙的なチェイニングは不可能である[1]．これは，メモリ上のベクトルデータに関するフロー依存関係を動的に解析するのが，極めて困難なことによる．一方，レジスタ-レジスタ演算方式の場合，フロー依存関係の解析対象はベクトル・レジスタであり，実行時でも容易に行える．つまり，ベクトル・レジスタを介した暗黙的なチェイニングが容易に実施できる．このチェイニングにより，レジスタ-レジスタ演算方式はメモリ-メモリ演算方式よりも（ピーク演算スループットが同一と仮定した場合）高い実効演算スループットを得ることができる．

以上の理由により，今日のスーパーコンピュータの大部分は，レジスタ-レジスタ演算方式を採用している．マイクロベクトルプロセッサ3機種のうち，NTT VPはメモリ-メモリ演算方式だが，日電VPPと富士通VPUはレジスタ-レジスタ演算方式を採用している．本稿では，以下，レジスタ-レジスタ演算方式のみを議論の対象とする．

[1] 専用命令による明示的なチェイニングは可能である．

表 1: 現在のマイクロベクトルプロセッサの仕様

マイクロベクトルプロセッサ	NTT VP	日電 VPP	富士通 VPU
製造プロセス	1μm CMOS	0.8μm 3 層 Al BiCMOS	0.5μm CMOS
動作周波数	40MHz	100MHz	70MHz
集積度	65KGate + 34Kbit	673K MOS-Tr + 18K Bi-Tr	1.5M Tr
ダイ・サイズ	?	17.2mm×17.3mm	15.75mm×16.00mm
I/O ピン数	?	528	256
ピーク性能	120MFLOPS(単精度)	200MFLOPS(倍精度)	289MFLOPS(単精度) 149MFLOPS(倍精度)
演算パイプライン数 (並列動作可能)	Add×2 Mul×1	Add×1 Mul/Div×1	Add×1 Mul×1 Div×1
ロード/ストア・ パイプライン数	Load×1 Load/Store×2	Load×1 Store×1	Load/Store×1
メモリ・バンド巾	480MB/秒	1600MB/秒	560MB/秒
ベクトル・ レジスタ容量	0	5KB (64 倍語×4 + 96 倍語×4)	8KB (256 倍語×4)

表 2: 演算スループットおよびメモリ・バンド巾

マシン	演算スループット			メモリ・バンド巾			
	# of Add/Mul Pipes	Throughput per Pipe†	Total Throughput†	# of Load/ Store Pipes	Bandwidth per Pipe‡	Total Load Bandwidth‡	Total Bandwidth‡
(1) 従来のスーパーコンピュータ：							
Cray X-MP Cray Y-MP	Add×1 Mul×1	1 1	2	Load×2 Store×1	1 1	2	3
富士通 VP-200	Add×1 Mul×1	2 2	4	Load/Store×2	2	4	4
日立 S-810/20	Add×2 Mul&Add×2	2 2	8	Load×3 Load/Store×1	2 2	8	8
日立 S-820/80	Add×1 Mul&Add×1	4 4	8	Load×1 Load/Store×1	4 4	8	8
日電 SX-2	Add×1 Mul×1	4 4	8	Load×1 Store×1	8 4	8	8
日電 SX-3	Add×2 Mul×2	4 4	16	Load×2 Store×1	4 4	8	12
(2) 現在のマイクロベクトルプロセッサ：							
NTT VP	Add×2 Mul×1	1§ 1§	2§	Load×1 Load/Store×2	1¶ 1¶	2¶	3¶
日電 VPP	Add×1 Mul×1	1 1	2	Load×1 Store×1	1 1	1	2
富士通 VPU	Add×1 Mul×1	1 1	2	Load/Store×1	1	1	1

†: 倍精度浮動小数点演算/クロック・サイクル
‡: 64 ビット倍語/クロック・サイクル
§: 単精度浮動小数点演算/クロック・サイクル
¶: 32 ビット語/クロック・サイクル

2.2 メモリ・バンド巾

前節で述べたように，所望の実効演算スループット (ST) を一定と仮定した場合，レジスタ-レジスタ演算方式に要求される要求メモリ・バンド巾 (MB_{RR}) は，メモリ-メモリ演算方式に対するそれ (MB_{SS}) よりも下式の通り小さい．

$$ST \times 3 \leq MB_{SS}$$
$$ST \leq MB_{RR} \leq ST \times 3$$

しかしながら，レジスタ-レジスタ演算方式でも，メモリ・バンド巾を高めざるを得ない要因が存在する．たとえば，元のベクトル長がベクトル・レジスタ長よりも長い場合に行うストリップ・マイニング (*strip mining*) 操作がそうである．これは，ベクトル・レジスタ長をセグメント長としてベクトルのセグメント化を行い，セグメント単位でベクトル演算を繰り返す手法である．このとき，セグメント単位でベクトル演算を一時中断し，現セグメントの結果ベクトルのストア，および，次セグメントのソース・ベクトルのロードが必要となり，そのオーバヘッドが問題となる．

結局のところ，レジスタ-レジスタ演算方式において，メモリ-メモリ演算方式ほどではないにしても，そ

表 3: マイクロプロセッサ技術の推移

技術	過去 10 年間	年成長率	1992 年現在	2000 年予想
動作周波数	7 倍	22%	50MHz	250MHz
性能	35 倍	43%	50MIPS	1000MIPS
トランジスタ数	30 倍	41%	1,500,000	50,000,000
チップ面積	5 倍	17%	$200mm^2$	$1000mm^2$
設計ルール	0.3 倍	−12%	$0.8\mu m$	$0.25\mu m$

こそこのメモリ・バンド巾が必要となる．表 2-(1) に示すように，従来のスーパーコンピュータでは，ピーク演算スループットとほぼ同程度のメモリ・バンド巾を提供している．すなわち，

ピーク・メモリ・バンド巾$_{RR}$ ≃ ピーク演算スループット

というのが実情である．一方，現在のレジスタ-レジスタ演算方式のマイクロベクトルプロセッサ 2 機種を見てみると（表 2-(2) 参照），日電 VPP は演算スループットとメモリ・バンド巾が等しいが，富士通 VPU の方はメモリ・バンド巾が演算スループットの 1/2 にしか満たないことが判る．

マイクロベクトルプロセッサは，チップ内ピーク演算スループットに見合っただけのチップ外ピーク・メモリ・バンド巾を今後とも提供できるのであろうか？

表 3 から判るように，集積度の伸び（年率 41%）に追従して演算パイプライン数を増やし，ピーク演算スループットを向上させることは可能である．しかしながら，集積度の伸びに比べてチップ 1 辺の長さの伸び（面積の伸びで年率 17%）は大きくなく，入出力バッド数およびピン数はそれほど伸びそうにない．すなわち，ピン・ボトルネックにより実効的な演算スループットが低く抑えられる危険性がある．

2.3 メモリ・バンド巾節約策

前節で提示した問題の解決のためには，従来のレジスタ-レジスタ演算方式にも増して，チップ外メモリ・バンド巾への要求を低く抑えることは重要である．そこで，チップ内レジスタ・バンド巾を活用し，その分だけチップ外メモリ・バンド巾への要求を削減可能な以下の 2 方式を検討する．

a. **FIFO ベクトル・レジスタ**（*F: FIFO*）：ベクトル・レジスタをリング・バッファ FIFO 動作させ，論理的に無限長のレジスタのように扱う．これにより，1 個のベクトル命令で一時に処理可能なベクトルの長さを制限する必要がなくなる．すなわち，ストリップ・マイニング操作が不要となり，これに伴うオーバヘッドを排除できる．

b. **ベクトル命令レベル・マルチスレッド処理**（*M: Multithreading at the vector-instruction level*）：ベクトル命令レベルでマルチスレッド処理を行う [5]．すなわち，1 本のパイプラインは，1 個のベクトル命令の実行に占有されるのではなく，複数個のベクトル命令に時分割共有される．このとき，個々のベクトル命令をディスパッチする対象であるパイプラインを仮想パイプライン（*virtual pipeline*）と，また，実在するパイプラインを実パイプライン（*real pipeline*）とそれぞれ呼ぶ．これにより，実効演算スループットを向上させると同時に，一時に実行可能なベクトル命令の数を増やせる．

仮想パイプラインを実パイプラインにディスパッチする方法（実パイプライン割当アルゴリズム）には次の 3 方式があるが，その選択に当たってはメモリ・アクセス・レイテンシ等を考慮に入れる必要がある [2]．

i. *Periodic*：切替え間隔およびディスパッチ間隔ともに一定かつ固定．たとえば，毎クロック・サイクル，仮想パイプラインを切替え（*cycle-by-cycle interleaving*），同一仮想パイプラインが実パイプラインにディスパッチされる間隔もコンパイラあるいはアーキテクチャにより静的に決まる．

ii. *Round-robin*：切替え間隔は一定かつ固定だが，ディスパッチ間隔は可変で動的に決まる．

iii. *Forced*：切替え間隔およびディスパッチ間隔ともに可変で動的に決まる．すなわち，仮想パイプラインの処理を阻止する要因が発生するまで，当該仮想パイプラインの処理を継続する．

メモリ・バンド巾の節約は，FIFO ベクトル・レジスタ，ベクトル命令レベル・マルチスレッド処理，および，チェイニング機能を組み合わせることで可能となる．また，ベクトル命令レベル・マルチスレッド処理とより柔軟なチェイニング機能[2]との組合せにより，従来では特殊なマクロ命令と専用ハードウェアなしではベクトル化不可能であったループが，一般命令のみでベクトル化可能となる [7]．

3 検討項目 2

3.1 命令-レジスタ対応関係

レジスタ-レジスタ演算方式の場合，スカラ・レジスタとベクトル・レジスタの 2 種類のレジスタが一般に存在する．また，命令としては，スカラ命令とベクトル命令の 2 種類がやはり存在する．したがって，どの命令がどのレジスタにアクセス可能とするかで，以下の 4 通りのアクセス機能が存在する．

- **SISR**（*Scalar Instructions – Scalar Registers*）機能：スカラ命令がスカラ・レジスタにアクセスして演算等を行う．通常のスカラ処理形態．

- **SIVR**（*Scalar Instructions – Vector Registers*）機能：スカラ命令がベクトル・レジスタにアクセスして演算等を行う．一般には，未活用．

[2]チェイニング方向として，従来から可能であった「先行命令→後続命令」に加えて，「後続命令→先行命令」および「命令→同一命令」も可能とする．

- **VISR** (*Vector Instructions – Scalar Registers*) 機能：ベクトル命令がスカラ・レジスタにアクセスして演算等を行う．従来のベクトル・プロセッサでも使用している．

- **VIVR** (*Vector Instructions – Vector Registers*) 機能：ベクトル命令がベクトル・レジスタにアクセスして演算等を行う．通常のベクトル処理形態．

2.3節で述べたように，ベクトル命令レベル・マルチスレッド処理とより柔軟なチェイニング機能の組合せにより，ベクトル化可能な範囲が拡大する．しかしながら，いずれにせよ，オブジェクト・コード中からスカラ実行部分がなくなることはない．*Amdahl*の法則が教える通り，プロセッサ全体の性能向上を図るためには，スカラ実行部分の性能向上が重要である．その方法の1つが，上記**SIVR**機能の活用である．すなわち，スカラ命令がベクトル・レジスタ内のベクトル・オペランドに対して直接アクセスして演算を行うことを許すのである．

3.2 命令セット・アーキテクチャ

レジスタ-レジスタ演算方式における命令セット（ISP: *Instruction-Set Processor*）アーキテクチャとして，以下の4種類が考えられる．

a.ISP_p (*purely scalar instruction set*)：すべての命令がスカラ命令として定義され，ベクトル命令は存在しない．スカラ命令はスカラ・レジスタ同様，ベクトル・レジスタにもアクセス可能である．つまり，前節で定義した**SISR**および**SIVR**機能を活用する．本アーキテクチャは，スーパースカラまたはVLIW(Very Long Instruction Word)にベクトル・レジスタを導入したものと見なせる．たとえば，directed-dataflowアーキテクチャ[12]やハイパースカラ・プロセッサ・アーキテクチャ[3]が本アーキテクチャに相当する．

b.ISP_c (*combined set of scalar arithmetic instructions and vector load/store instructions*)：すべての演算命令がスカラ命令として定義される．ベクトル演算命令は存在しないが，ベクトル・ロード／ストア命令は存在する．一連のベクトル処理は，次の手順で行われる．

i. ベクトル・ロード命令がベクトル・レジスタにベクトルデータをロードする（**VIVR**）．
ii. スカラ命令のループがベクトル・レジスタに直接アクセスして演算を施す（**SIVR**）．
iii. ベクトル・ストア命令が演算結果をベクトル・レジスタからメモリへストアする（**VIVR**）．

WMアーキテクチャ[13]が本アーキテクチャに相当する．

c.ISP_s (*separate scalar&vector instruction sets*)：従来のベクトル・プロセッサのほとんどが本アーキテクチャを採用している．つまり，スカラ命令セットとベクトル命令セットとは別個のものであり，スカラ命令はスカラ・レジスタのみに（**SISR**），ベクトル命令はスカラ・レジスタ（**VISR**）とベクトル・レジスタ（**VIVR**）にそれぞれアクセスできる．SIVR機能は活用されない．

d.ISP_u (*unified scalar/vector instruction set*)：すべての命令は「あるベクトル長を有するベクトル命令」として定義される．スカラ命令は定義されないが，これはベクトル長1のベクトル命令に相当する．よって，**SISR**，**SIVR**，**VISR**，および，**VIVR**の全機能を活用する．たとえば，MultiTitan[9]やMSFVアーキテクチャ[6, 7, 2]が本アーキテクチャに相当する．

4 評価方法

前2章で検討した項目について，ソフトウェア・シミュレータを用いて評価を行った．

まず，3章で述べた4つの命令セット・アーキテクチャのうち，次の3つを評価モデルとして設定した．

a.ISP_p (purely scalar ISP)[3]

b.ISP_s (separate scalar&vector ISP)

c.ISP_u (unified scalar/vector ISP)：さらに，ISP_uモデル上で，2章で述べた次のメモリ・バンド巾節約策も評価した．

- ISP_u+M：ISP_uモデルにベクトル命令レベル・マルチスレッド処理機能を追加．仮想演算パイプラインおよび仮想ロード／ストア・パイプラインをそれぞれ8本設ける．仮想演算パイプラインには，1本の実加算パイプラインと1本の実乗算パイプラインが対応する．また，仮想ロード／ストア・パイプラインには，2本の実ロード／ストア・パイプラインが対応する．実パイプライン割当アルゴリズムには，round-robin方式を用いる．
- ISP_u+M+F：上記ISP_u+Mモデルにさらに，FIFOベクトル・レジスタ機能を追加．

表4に各モデルの主な仕様を示す．

ベンチマーク・プログラムには，LFK（*Livermore Fortran Kernels*）24の中からLFK1～LFK14を用いる．評価指標には，正規化FLOPC（*normalized floating-point operations per clock cycle*）を用いる．なお，各評価モデルのピーク演算スループットは2倍精度浮動小数点演算/クロック・サイクルであるから，ピーク性能も2FLOPCとなる．

5 評価結果および考察

5.1 命令セット・アーキテクチャ

図1に，評価モデルISP_p，ISP_s，および，ISP_uに対する評価結果を示す．図1より次のことが判る．

- ISP_uモデル：3モデルの中で最も性能が高い．ISP_pと比べて，最低0.43倍～相乗平均1.80倍～最高5.28倍の性能である．また，ISP_sと比べて，最低1.00倍～相乗平均1.03倍～最高1.22倍の性能である．

- ISP_sモデル：ISP_uモデルに次いで性能が高い．ISP_pと比べて，最低0.43倍～相乗平均1.76倍～最大5.28倍の性能である．また，ISP_uと比べて，最低0.82倍～相乗平均0.98倍～最大1.00倍の性能である．

[3]今回の評価では，ISP_pはベクトル・レジスタを持たない通常のスーパースカラ・プロセッサをモデル化している．

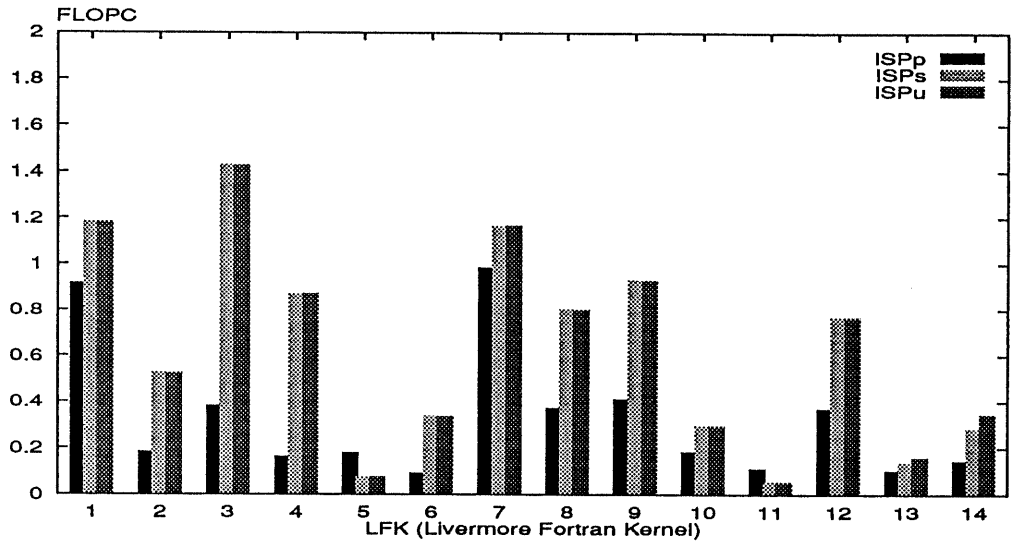

図 1: 評価モデル ISP_p, ISP_s, および, ISP_uの性能

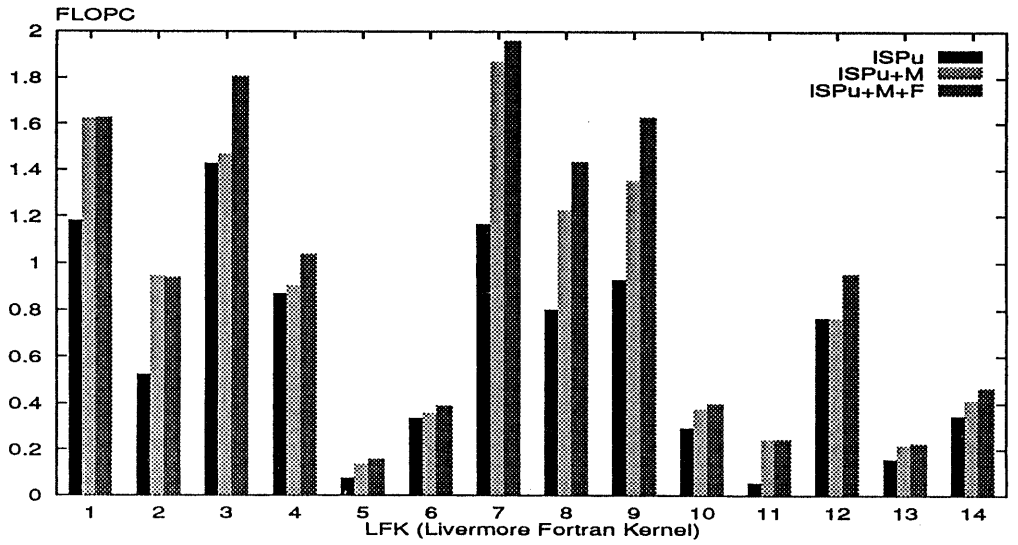

図 2: 評価モデル ISP_u, $ISP_u + M$, および, $ISP_u + M + F$の性能

- ISP_pモデル:3 モデルの中で最も性能が低い. ISP_sと比べて, 最低0.19 倍～相乗平均 0.59 倍～最大 2.32 倍の性能である. また, ISP_uと比べて, 最低0.19 倍～相乗平均0.55 倍～最大2.32 倍の性能である.

まず, 今回用いたISP_pモデルは, ベクトル・レジスタを持たない通常のスーパースカラ・プロセッサであり, 3章で定義したISP_pそのものとは異なる. 3章の定義通りに正確にISP_pをモデル化すれば, 性能はまだ向上するものと期待される. これは今後の課題である. しかし, それにも関わらず, LFK5 と LFK11 においては, ISP_pモデルがISP_sおよびISP_uに比べて, 2.32 倍 (LFK5) および2.02 倍 (LFK11) の性能となっている.

次に, 評価モデル ISP_sおよび ISP_uに着目する. ISP_uモデルは, **SIVR** (Scalar Instructions – Vector Registers) 機能を活用している点で ISP_sモデルと異なる. ベンチマーク・プログラム LFK1～14 においては, LFK13 と LFK14 のみが, **SIVR** 機能の適用が可能である. 図 3に, LFK13 における **SIVR** 機能の適用の様子を示す.

LFK13 の配列 H は, その添字が再び配列 E および F で与えられる間接添字の形になっている. したがって, 配列 H に関するデータ依存関係の有無がコンパ

表 4: 評価モデルの仕様

評価モデル		ISP_p	ISP_s	ISP_u
命令発行バンド巾♮		4		1
Add/Mul パイプライン	# of Pipes	Add×1, Mul×1		
	Issue Latency†	1		
	Result Latency†	4	Add: 5, Mul: 6	
	Throughput per Pipe‡	1		
	Total Throughput‡	2		
Load/Store パイプライン	# of Pipes	Load/Store×2		
	Issue Latency†	1		
	Access Latency†	2	6	
	Bandwidth per Pipe§	1		
	Total Bandwidth§	2		
スカラレジスタ	# of Registers	General-purpose: 32		
		Floating-point: 32		
ベクトルレジスタ	# of Registers	0	16	
	Register Length	—	32	
	Bandwidth per Register§	—	1	
メモリ	# of Ports	2		
	Port Size♯	8		
	# of Banks	2	8	
	Bank Size♯	8		
	Interleaving Width♯	8		
	Bank-Busy Time†	1		
	Bandwidth per Port§	1		

♮: 命令/クロック・サイクル
†: クロック・サイクル
‡: 倍精度浮動小数点演算/クロック・サイクル
§: 64 ビット倍語/クロック・サイクル
♯: バイト

イル時に断定できないので，LFK13 の元の DO ループ 13 全体をベクトル化することは出来ない．そこで，図3の下部に示すように，DO ループ 13 と DO ループ 131 の 2 つにループ分割する．さらに，スカラ・エクスパンションも施す．この結果，DO ループ 131 はベクトル化不可能なままだが，DO ループ 13 はベクトル化可能となった．

ここで，DO ループ 131 は DO ループ 13 に対して，配列 TI23 および TJ23 に関してフロー依存関係にある．ここに，**SIVR** 機能を適用する．すなわち，ISP_s モデルでは，配列 TI23 および TJ23 をベクトル・ストア命令で一旦メモリにストアした後，DO ループ 131 中のスカラ・ロード命令で配列要素 TI23(IP) および TJ23(IP) を再ロードしなければならなかった．これに対して，ISP_u では **SIVR** 機能により，DO ループ 131 中のスカラ命令が，ベクトル・レジスタを介して配列 TI23 および TJ23 に直接アクセスできる．これにより，ロード／ストア回数が削減できるだけでなく，ベクトル実行部分とスカラ実行部分との間のオーバラップ実行が行えるようになる．

図 1において，**SIVR** 機能による ISP_u モデルの ISP_p モデルに対する性能向上率は，LFK13 で 17%，LFK14 で 22%となっている．

5.2 ベクトル命令レベル・マルチスレッド処理およびFIFO ベクトル・レジスタ

図 2に，評価モデル ISP_u，$ISP_u + M$，および，$ISP_u + M + F$に対する評価結果を示す．図 2より次のことが判る．

```
      DO 13 IP= 1,N
         I1= P(1,IP)
         J1= P(2,IP)
         I1= 1 + MOD2N(I1,64)
         J1= 1 + MOD2N(J1,64)
         P(3,IP)= P(3,IP) + B(I1,J1)
         P(4,IP)= P(4,IP) + C(I1,J1)
         P(1,IP)= P(1,IP) + P(3,IP)
         P(2,IP)= P(2,IP) + P(4,IP)
         I2= P(1,IP)
         J2= P(2,IP)
         I2= MOD2N(I2,64)
         J2= MOD2N(J2,64)
         P(1,IP)= P(1,IP) + Y(I2+32)
         P(2,IP)= P(2,IP) + Z(J2+32)
         I2= I2 + E(I2+32)
         J2= J2 + F(J2+32)
         H(I2,J2)= H(I2,J2) + 1.0D0
   13    CONTINUE

      ⇓ ループ分割およびスカラ・エクスパンション

      DO 13 IP= 1,N
         TI11(IP)= P(1,IP)
         TJ11(IP)= P(2,IP)
         TI12(IP)= 1 + MOD2N(TI11(IP),64)
         TJ12(IP)= 1 + MOD2N(TJ11(IP),64)
         P(3,IP)= P(3,IP) + B(TI12(IP),TJ12(IP))
         P(4,IP)= P(4,IP) + C(TI12(IP),TJ12(IP))
         P(1,IP)= P(1,IP) + P(3,IP)
         P(2,IP)= P(2,IP) + P(4,IP)
         TI21(IP)= P(1,IP)
         TJ21(IP)= P(2,IP)
         TI22(IP)= MOD2N(TI21(IP),64)
         TJ22(IP)= MOD2N(TJ21(IP),64)
         P(1,IP)= P(1,IP) + Y(TI22(IP)+32)
         P(2,IP)= P(2,IP) + Z(TJ22(IP)+32)
         TI23(IP)= TI22(IP) + E(TI22(IP)+32)
         TJ23(IP)= TJ22(IP) + F(TJ22(IP)+32)
   13    CONTINUE
         DO 131 IP= 1,N
            H(TI23(IP),TJ23(IP)) =
                  H(TI23(IP),TJ23(IP)) + 1.0D0
   131   CONTINUE
```

図 3: LFK13 (2-D Particle in Cell)

- ベクトル命令レベル・マルチスレッド処理による（$ISP_u + M$モデルの ISP_uモデルに対する）性能向上率は，最低0.0%～相乗平均44%～最高334%とかなり大きい．性能向上率が低いのは LFK3 (2.9%) と LFK12 (0.0%) であるが，これらのループは演算数自体が少なく（LFK3 は乗算 1 と加算 1，LFK12 は減算 1），ベクトル命令レベル・マルチスレッド処理の効果が得られないためである．逆に，演算数が多いループに対しては，ベクトル命令レベル・マルチスレッド処理の効果は大きい．とりわけ，演算数が多く，かつ，回帰型演算（一次回帰演算，総和，内積，等）を含むような場合，その効果は顕著である．たとえば，LFK11 (first summation) ではベクトル命令レベル・マルチスレッド処理により 4 倍以上もの性能が得られている．

- FIFO ベクトル・レジスタによる（$ISP_u + M + F$ モデルの $ISP_u + M$モデルに対する）性能向上率は，最低0.0%～相乗平均11%～最高24%である．この性能向上は，ストリップ・マイニング操作に伴うオーバヘッドが除去されたことにより得られたものである．ただし，FIFO ベクトル・レジスタに要するコストはかなり大きく，本機能を導入するか否かに関しては性能とハードウェア・コス

トとのトレードオフに十分留意して決定する必要がある [2].

6 おわりに

以上，マイクロベクトルプロセッサ構築に適したベクトル・アーキテクチャについて検討した．マイクロベクトルプロセッサは，I/O ピン数の制約によりチップ外メモリ・バンド巾を大幅に節約しなければならない．さらに，従来にも増して，ベクトル化可能範囲を拡大し，かつ，スカラ実行部分の性能を向上させる必要がある．本稿では，このようなマイクロベクトルプロセッサに有効なアーキテクチャ機能として，FIFO ベクトル・レジスタ，ベクトル命令レベル・マルチスレッド処理，および，4 種類の命令セット・アーキテクチャ（ISP_p, ISP_c, ISP_s, ISP_u）を提案し，その効果をシミュレーションにより評価した．

まず，命令セット・アーキテクチャ（ISP_p, ISP_s, ISP_u）に関して，以下の結果を得た．

- ISP_u モデル:3 モデルの中で最も性能が高い．ISP_p と比べて，最低 0.43 倍～相乗平均 1.80 倍～最高 5.28 倍の性能である．また，ISP_s と比べて，最低 1.00 倍～相乗平均 1.03 倍～最高 1.22 倍の性能である．

- ISP_s モデル：ISP_u モデルに次いで性能が高い．ISP_p と比べて，最低 0.43 倍～相乗平均 1.76 倍～最大 5.28 倍の性能である．また，ISP_u と比べて，最低 0.82 倍～相乗平均 0.98 倍～最大 1.00 倍の性能である．

- ISP_p モデル:3 モデルの中で最も性能が低い．ISP_s と比べて，最低 0.19 倍～相乗平均 0.59 倍～最大 2.32 倍の性能である．また，ISP_u と比べて，最低 0.19 倍～相乗平均 0.55 倍～最大 2.32 倍の性能である．

また，ISP_u モデル上で，ベクトル命令レベル・マルチスレッド処理，および，FIFO ベクトル・レジスタについて評価を行い，次の結果を得た．

- ベクトル命令レベル・マルチスレッド処理により，最低 0.0%～相乗平均 44%～最高 334%の性能向上が得られた．

- FIFO ベクトル・レジスタにより，最低 0.0%～相乗平均 11%～最高 24% の性能向上が得られた．

以上の評価結果より，ISP_u の **SIVR** 機能，および，ベクトル命令レベル・マルチスレッド処理は，マイクロベクトルプロセッサ・アーキテクチャとして適切な機能だと言えよう．なお，FIFO ベクトル・レジスタに関しては，そのハードウェア・コストはかなり大きく，本機能を導入するか否かに関しては性能とコストとのトレードオフに十分留意して決定する必要がある．

謝辞

日頃ご討論頂く九州大学 大学院総合理工学研究科 安浦研究室の諸氏に感謝致します．

参考文献

[1]位守, 伊藤, 中村, 中澤, “擬似ベクトル処理向きメモリアーキテクチャの一提案,” 情処研報, ARC-91-8, 1991 年 11 月.

[2]橋本, 岡崎, 弘中, 村上, 富田, “『順風』: MSF 型ベクトル・プロセッサ・プロトタイプ— MSFV アーキテクチャに関する評価 —,” 並列処理シンポジウム *JSPP'92* 論文集, 1992 年 6 月.

[3]村上和彰, “ハイパースカラ・プロセッサ・アーキテクチャ— 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.101–108, 1991 年 5 月.

[4]村上和彰, “マイクロプロセッサ・アーキテクチャと並列処理技術— マイクロプロセッサ用並列処理と並列処理用マイクロプロセッサ —,” 信学技報, ICD-91-91, 1991 年 9 月.

[5]Chiueh, T.-C., “Multi-Threaded Vectorization,” *Proc. 18th Int'l. Symp. Computer Architecture*, pp.352–361, May 1991.

[6]Hironaka, T., Hashimoto, T., Okazaki, K., Murakami, K., and Tomita, S., “A Single-Chip Vector-Processor Prototype Based on Multithreaded Streaming/FIFO Vector (MSFV) Architecture,” *Proc. Int'l. Symp. Supercomputing'91*, pp.77–86, Nov. 1991.

[7]Hironaka, T., Hashimoto, T, Okazaki, K., Murakami, K., and Tomita, S., “Benchmarking a Vector-Processor Prototype Based on Multithreaded Streaming/FIFO Vector (MSFV) Architecture,” to appear in *Proc. 1992 Int'l. Conf. Supercomputing*, July 1992.

[8]Iino, H., Takahashi, H., Sukemura, T., Kimura, M., Fujita, K., and Mori, S., “A 289MFLOPS Single-Chip Supercomputer,” *Digest of Technical Papers, 1992 IEEE Int'l. Solid-State Circuits Conf.*, TA 6.5, Feb. 1992.

[9]Jouppi, N. P., Bertoni, J., and Wall, D. W., “A Unified Vector/Scalar Floating-Point Architecture,” *Proc. 3rd Int'l. Conf. Architectural Support for Programming Languages and Operating Systems*, pp.134–143, Apr. 1989.

[10]Nakada, H., Sakurai, N., Kanayama, Y., Ohta, N., and Oguri, K., “Vector Processor Design for Parallel DSP Systems Using Hierarchical Behavioral Description Based Synthesizer,” *Proc. 1990 IEEE Int'l. Conf. Computer Design: VLSI in Computers & Processors*, pp.86–89, Sept. 1990.

[11]Okamoto, F., Hagihara, Y., Ohkubo, C., Sekine, Y., Nishi, N., Yamada, H., and Enomoto, T., “A 200MFLOPS 100MHz 64b BiCMOS Vector Pipelined Processor,” *Digest of Technical Papers, 1991 IEEE Int'l. Solid State Circuits Conf.*, FPM15.5, pp.256–257, Feb. 1991.

[12]Rau, B. R., Yen, D. W. L., Yen, W., and Towle, R. A., “The Cydra 5 Departmental Supercomputer: Design Philosophies, Decisions, and Trade-offs,” *Computer*, vol.22, no.1, pp.12–35, Jan. 1989.

[13]Wulf, W. A., “The WM Computer Architecture, ” *ACM SIGARCH Computer Architecture News*, vol.16, no.1, pp.70–84, ar. 1988.