■6 群（コンピュータ －基礎理論とハードウェア）－ 5 編（コンピュータアーキテクチャ（II）先進的）

4 章 ベクトルコンピュータ

（執筆者：佐藤寿倫）［2010 年 5 月 受領］

■概要■

ベクトル方式はかつてのスパコンで利用された技術であるが，形を変えて現在の先端プロセッサの中で生き続けている．本章では，ベクトル方式の有効性が高いアプリケーションの紹介から話を始め，形を変えて現在の商用コンピュータでも利用されているベクトル方式を解説し，ベクトル方式の利用拡大にとって将来有望な選択肢のひとつを紹介する．

【本章の構成】

現在の主要なアプリケーションであるマルチメディア処理はベクトル方式と親和性が高い．4-1 節でそれを解説する．続いて 4-2 節で，現在の主要なベクトルコンピュータである SIMD 型命令コンピュータを解説する．最後に，SIMD 型命令の利用を広げるために検討されている動的ベクトル化方式を，4-3 節で紹介する．

 © 電子情報通信学会 *2010*

■6 群 － 5 編 － 4 章

4-1 マルチメディア処理とベクトルコンピュータ

（執筆者：平澤将一）［2009 年 1 月 受領］

4-1-1 ベクトルコンピュータの誕生

ベクトルコンピュータは，1 個の命令を用いて複数のデータが含まれるベクトルに対する演算を多数の ALU で同時に行うことが可能なコンピュータアーキテクチャである．ベクトルは，メモリ中の配列やベクトルレジスタに保存されたデータ列である．

配列やベクトルレジスタを同時に演算する命令を記述することで，これらデータ構造に含まれる各データ要素間にはデータ依存性がないことが保証される．したがって，複数の ALU を用いて並列に演算を行うことが可能である．また，ベクトル演算中は規則的なベクトルデータの演算を行うため制御の分岐がなく，深いパイプラインによる高性能化が容易である．以上から，ベクトルコンピュータでは理論的に高いピーク演算性能が得られる．

高いピーク演算性能を生かして高い実効性能を得るためには，十分なメモリバンド幅により ALU にデータを供給し続ける必要がある．ベクトルコンピュータにおいてはデータが規則的なデータ構造に保存されているためメモリアクセスが規則的となることが多く，メインメモリのマルチバンク化で高いメモリバンド幅を得られる．また，データ数に対して少数の命令で実行を行うことからフェッチする命令数がスーパースカラプロセッサと比較して少なく，高いデータ転送レートを保つことが容易である．

その他，実行する命令数が少ないことから命令のフェッチ及びデコード数が演算量と比較して少なく，ベクトル演算を実行している間は制御の分岐もないためハードウェアにかかる負担が小さく，プロセッサ制作が容易で高いクロック周波数を得やすい特徴がある．

以上から，ベクトルコンピュータは高いデータ転送レートを用いて大量のデータを ALU に供給して高い並列度で計算し，高い実効性能を得ることが可能なコンピュータアーキテクチャである．

4-1-2 ベクトルコンピュータの発展

ベクトルコンピュータは,主に大規模データを扱う高性能科学技術計算において発展した．アプリケーションとしては天気予報や，自動車衝突などのシミュレーションがある．このようなアプリケーションは大量のデータを繰り返し計算することが多く，キャッシュを用いたスカラプロセッサには不向きである．

ベクトルプロセッサのアーキテクチャとして，メモリに格納された配列を対象として演算を行うメモリ-メモリベクトルマシンと，ベクトルレジスタを備え，メモリからのベクトルロードやストアを行ってベクトルレジスタを対象にベクトル演算を行うベクトルレジスタマシンがある．スカラプロセッサにおいてレジスタを対象に演算を行うアーキテクチャが主流となっているのと同様，ベクトルプロセッサにおいてもベクトルレジスタマシンが主流となっている．

高性能コンピュータの LINPACK 実行性能のリストである TOP 500 List[1) において,かつてはベクトルプロセッサを用いたベクトルコンピュータが大勢を占めていた.しかし現在では,スーパースカラプロセッサによる汎用 PC クラスタが増加してきている．2002 年から 2004

 © 電子情報通信学会 2010

年にかけてNEC製のベクトル型スーパーコンピュータ地球シミュレータが1位となるのを最後に，ベクトルコンピュータがトップとはなっていない．

4-1-3 アプリケーションの変化，スーパースカラプロセッサの台頭

一般用 PC の普及によりスーパースカラプロセッサが普及している．大量の出荷数によりプロセスが改善され，高いクロック，高い性能を実現している．またコストの削減も進み，ベクトルプロセッサの価格競争力が相対的に減少していった．

一方，高性能科学技術計算だけでなく，画像，音声，動画を扱うマルチメディアアプリケーションが一般にも広く用いられるようになってきた．これらのアプリケーションを高性能に実行できるよう，スーパースカラプロセッサにおいても単一の命令により短い（4 ワード幅などの）ベクトルレジスタを用いて実行を行うことができる SIMD 命令が搭載されはじめ，広く用いられるようになっている．

近年では更に，性能辺りの消費電力量が少なく抑えられるという特徴から，高性能コンピューティング分野だけでなく，特に消費電力の制約が厳しく，またマルチメディアアプリケーションが重要である携帯電話などの組込み用途においては SIMD 命令が広く普及し，活用されている．

4-1-4 まとめ

ベクトルコンピュータにおいてもベクトルキャッシュなどスーパースカラプロセッサで用いられている技術が取り入れられており，ベクトルプロセッサとスーパースカラプロセッサがアーキテクチャとしてお互いに接近していく現象も見られる．

近年では，多数の演算コアをもった GPU も台頭してきている．GPU は，元来画像処理を専門に処理するプロセッサとして大量のデータを並列に計算するベクトル的アーキテクチャであり，プログラミングの汎用性をもつに至り，高性能科学技術計算分野で普及してきている．

以上から，スーパーコンピュータとしての典型的ベクトルコンピュータはその数を減らしつつあるが，多数の規則的データを少数の命令を用いて並列に実行することで高性能，低消費電力を達成するというそのコンセプトはますます広がりを見せているといえる．

■参考文献

1) TOP 500 List. http://www.top500.org/

■6 群 － 5 編 － 4 章

4-2 SIMD 型命令コンピュータ

（執筆者：林　宏雄）［2008 年 10 月 受領］

本節では SIMD 型命令コンピュータの定義と，一般的な実現例として SIMD 型マイクロプロセッサについて解説する．

4-2-1 SIMD 型命令コンピュータ

コンピュータシステムは，その命令列及びデータ列の並列性に基づいて，SISD（Single Instruction stream, Single Data stream），SIMD（Single Instruction stream, Multiple Data stream），MISD（Multiple Instruction stream, Single Data stream），MIMD（Multiple Instruction stream, Multiple Data stream）の四つに分類される [1]．このうち，SIMD は一つの命令列を複数の実行ユニットで実行するものである．複数のデータに対して並列に演算を行うデータ並列性をもつ処理に対して有効であり，一つの命令制御回路で，演算器の分割，もしくは追加することで実装することが可能である．前節で説明されたベクトルコンピュータや，多くの DSP（Digital Signal Processor）も SIMD 型に分類される．

4-2-2 マイクロプロセッサにおける SIMD 命令拡張

マイクロプロセッサは一般に，一つの命令ストリームを読み込み，同時に一つのデータを処理する．メディア，通信，グラフィックなどの処理では，演算器のデータ幅よりも小さい 8 bit，16 bit，もしくは 32 bit などのデータの演算を行う．これらのデータを複数レジスタに格納し（パックドデータ形式などと呼ばれる），一つの命令でそれぞれの要素（element）に対して並列に処理を行う SIMD 型命令と呼ばれる命令が提案された．商用汎用マイクロプロセッサでは，Intel 社の i860（1989 年），Motorola 社の MC 88110（1991 年）以降，主要なマイクロプロセッサに導入された（**表 4･1**）．

表 4・1　主要 RISC マイクロプロセッサの SIMD 命令拡張

アーキテクチャ	HP PA-RISC	Sun SPARC	MIPS	DEC Alpha	PowerPC	ARM
SIMD 拡張名	MAX	VIS	MDMX	MVI	AltiVec/VMX	NEON
実　装	1994 年 7100 LC	1995 年 Ultra SPARC	none	1997 年 21164 PC	1999 年 G4	2005 年 ARM v7
レジスタファイル	64 bit×32 整数共有	64 bit×32 FP 共有	64 bit×32 FP 共有	64 bit×32 整数共有	128 bit×32 専用	128 bit×16 /64 bit×32 専用

4-2-3 x86（IA-32）アーキテクチャにおける SIMD 命令拡張

インテル IA-32 アーキテクチャ [2), 3)] の SIMD 命令は，1997 年の MMX 拡張に始まり，その後継続して拡張が行われ，2008 年の SSE 4.2 までに合計 370 種類もの命令が定義されている（**表 4･2**）．本項では，それぞれの拡張について解説することにより，SIMD 命令の具体例及び技術動向を示す．なお，これらの命令拡張では SIMD 命令以外の命令も含まれているが，本解説では原則として取り扱わない．

表 4・2　x86（IA-32）アーキテクチャにおける SIMD 命令拡張

SIMD 拡張名	年	実　装	命令数	レジスタファイル		主な拡張
Intel MMX	1997	Pentium (P55C)	57	64 bit×8	FP 共有	64 bit 整数 SIMD
AMD 3DNow!	1998	K6-2	21			単精度浮動小数点，SIMD
Intel SSE	1999	Pentium III (Katmai)	70	128 bit×8	専用	単精度浮動小数点 SIMD
Intel SSE 2	2001	Pentium 4 (Willamette)	144			倍精度浮動小数点 SIMD，128 bit 整数 SIMD
Intel SSE 3	2004	Pentium 4 (Prescott)	13			水平演算，非対称演算
Intel SSSE 3	2006	Core 2 (Merom)	32			SSE3 拡張の整数演算対応
Intel SSE 4.1	2007	Core 2 (Penryn)	47			既存命令直交化，特定応用対応
Intel SSE 4.2	2008	Core i7 (Nehalem)	7			既存命令直交化，特定応用対応

(1)　MMX

64 bit のレジスタが 8 本 MMX レジスタとして定義されたが，物理的には FPU のレジスタと共用されている．このため MMX 命令を FPU 命令と混在して使用することはできないが，コンテクスト切り替え時に FPU レジスタを保存する従来のオペレーティングシステムを変更することなく使用することが可能である．

新しいデータ型として，64 bit パックドバイト整数（符号付き／符号なし），64 bit パックドワード整数（符号付き／符号なし），64 bit パックドダブルワード整数（符号付き／符号なし）の 3 種類が定義されている．

MMX 拡張は，整数算術演算，比較，変換，アンパック命令，論理演算，シフト，データ転送，ステート制御を行う 57 の命令から構成される．このうち算術演算命令は，整数オーバー（アンダ）フロー時の挙動によって，ラップアラウンド算術，符号付き飽和算術，符号なし飽和算術の，3 種類に分類される．飽和演算は，オーバーフロー，アンダーフローが起きたときに，それぞれ最大値，最小値に丸めるもので，音声，画像データ処理などに用いられる．

(2)　3DNow!

AMD 社による拡張で，3 次元グラフィックスの座標計算の高速化などを目的とした，2 並列の 32 bit 単精度浮動小数演算のサポートが最も大きな特徴である．レジスタは従来の 64 bit MMX レジスタを使用する．

(3)　SSE

8 本の 128 bit XMM レジスタが追加された．オペレーティングシステムの対応が必要だが，命令当たり従来の 2 倍の演算が可能となり，FPU 演算との混在も可能となった．命令の拡張では 4 並列の 32 bit 単精度浮動小数演算のサポートが最大の特徴である．

SSE 命令は，パックド及びスカラ単精度浮動小数点命令，64 ビット SIMD 整数命令，ステート管理命令，及びキャッシュ制御命令／プリフェッチ命令／メモリアクセス順序命令の四つの機能グループに分類される．

パックド及びスカラ単精度浮動小数点命令は，データ転送命令，算術演算命令，論理演算命令，比較命令，シャッフル命令，及び変換命令からなる．パックド演算が 4 組の SIMD 演算を行うのに対して，スカラ演算は最下位ダブルワードのみの操作が行われる（**図 4･1**）.

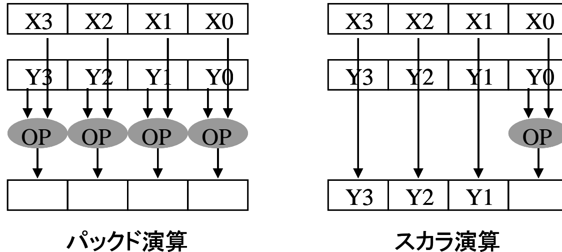

図 4・1　パックド演算とスカラ演算

データ転送命令は，XMM レジスタ同士の間もしくは XMM レジスタとメモリの間で，単精度浮動小数点データの転送を行う．算術演算命令は，加算，減算，乗算，除算，逆数計算，平方根計算，平方根の逆数計算，最大値/最小値計算を行う．論理演算命令は，AND，AND NOT，OR，及び XOR 演算を実行する．変換命令は，単精度浮動小数点フォーマットとダブルワード整数フォーマットの間で，パックド変換及びスカラ変換を実行する．64 ビット SIMD 整数命令は，MMX レジスタに対して操作を行うものである．平均，抽出（extract），挿入（insert），最大，最小，差分絶対値の合計，マスク，シャッフルからなる．

(4)　SSE2

倍精度浮動小数点演算，128 bit 整数 SIMD 演算の追加が主な拡張である．

x86 アーキテクチャでは，浮動小数点演算に 8087 以来スタックアーキテクチャを採用しており，浮動小数点レジスタを使う RISC アーキテクチャに対して性能上見劣りをしていた．SSE 2 ではレジスタベースのパックド及びスカラ倍精度浮動小数点演算命令を追加することにより，浮動小数点演算性能が大幅に向上した．

(5)　AMD 64，Intel 64

2003 年，AMD によって AMD 64 と呼ばれる 64 bit モード拡張が行われた（後に Intel も同じ命令拡張を踏襲することとなり，Intel 64 と命名された）．64 bit モードでは命令プレフィックスを用いることで，従来よりも倍の数のレジスタが使用できるようになり，合計 16 本の MXX レジスタにアクセス可能となった（MMX レジスタは 8 本のままである）．

(6)　SSE 3

非アライン 128 bit ロード命令，データ複製 SIMD 浮動小数点命令，非対称浮動小数点加減算命令，水平浮動小数点加減算命令，x87 整数返還命令，スレッド同期命令からなる．

従来のほぼすべての SIMD 演算が垂直演算であったのに対し，水平演算及び非対称垂直演算の追加が特徴的である．垂直演算（**図 4･2**(1)）は，要素ごと（intra-element）演算とも呼ばれ，被演算データの対応する要素ごとに演算を行う．要素ごとに独立に演算を行うため，要

素をまたがるデータパスが不要，他ビットの桁上がりが不要など，実装コストが小さく，すべての SIMD 拡張によってサポートされている．これに対して，水平演算（図 4･2(2)）は，要素間算術（inter-element arithmetic）演算とも呼ばれ，一つの被演算データの複数の要素間の演算を行う．非対称垂直演算（図 4･2(3)）は，垂直演算の一種であり，通常の垂直演算がすべての要素に対して同じ演算を行うのに対して，加算と減算など要素によって異なる演算を行う．

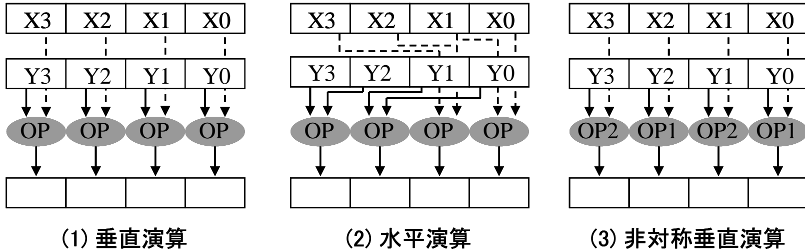

図 4・2　垂直演算，水平演算，非対称垂直演算

(7)　SSSE 3（Supplemental SSE 3）

SSE 3 で行われた拡張の整数演算対応が主な拡張である．

(8)　SSE 4

2007 年に Intel Core 2 で追加された SSE 4.1（47 命令）と，2008 年に Intel Core i7 で追加された SSE4.2（7 命令）の二組の拡張からなる．既存命令の直交化と，文字列処理命令など特定の応用に特化した命令の追加が行われている．

4-2-4 SIMD 型プロセッサ

一般のマイクロプロセッサでは,既存の命令セットに SIMD 型命令が追加拡張されている．これに対し，Cell Broadband Engine™（Cell/B.E.™）プロセッサ [4] の SPE（Synergistic Processor Element）[5), 6)] は，すべての算術・論理命令を SIMD 型で定義した SIMD 型プロセッサである．レジスタファイルは 128 bit 幅，128 本の構成で，すべての演算（整数，論理，浮動小数点）で使用される．128 bit のレジスタは 8 bit，16 bit，32 bit，64 bit に分割されて SIMD 演算を行う．

■参考文献

1)　M. J. Flynn, “Very high-speed computing systems,” Proc. IEEE 54:12 (December), 1901-0901.
2)　Intel Corp., “IA-32 インテルアーキテクチャー・ソフトウェア・デベロッパーズ・マニュアル, 上巻: 基本アーキテクチャー,” P9-1-P12-10, 2004.
3)　Intel Corp., “Intel 64 and IA-32 Architectures Software Developer7s Manual Volume1: Basic Architecture,” P9-1-P12-10, 2007.
4)　Dac C. Pham, et al., “The Design and Implementation of a First-Generation CELL Processor,” ISSCC 2005, pp.184-185.
5)　B. Flachs, et al., “A Streaming Processing Unit for a CELL Processor,” ISSCC 2005, pp.134-135.
6)　SCEI, Toshiba, IBM, “Synergistic Processor Unit Instruction Set Architecture Version 1.2,” 2007.

■6 群 － 5 編 － 4 章

4-3 動的ベクトル化方式

（執筆者：佐藤寿倫）［2008 年 10 月 受領］

近年の主流であるマルチメディア処理にはベクトル長の短いベクトル処理が適しており（本章 4-1 参照），多くのマイクロプロセッサでは命令セットを拡張して SIMD 命令を追加している（本章 4-2 参照）．残念ながら，それらは既存のハードウェアに無理矢理 SIMD 演算機能を追加したアドホックな拡張であり，その利用には大きなハードルがある．まず第 1 に，SIMD 命令を有効に利用できるコンパイラ技術が未成熟である．これは，既存のスカラ演算用のレジスタに SIMD 演算を割り当てるといった拡張が原因と考えられる．そのため，多くの場合，SIMD 演算ライブラリを利用するなどしてプログラマが明示的に SIMD 命令を利用しなければならない．第 2 に，SIMD 命令拡張の世代間に互換性がない．例えば，インテルは命令セットに MMX や SSE といった拡張を施してきたが，その拡張はハードウェア実装に依存し，世代ごとに命令を追加し続けなければならない．

本節では，これらの問題に対処する動的ベクトル化方式を紹介する．まず，コンパイラやプログラマの負担を軽減するために，プログラムの実行時に SIMD 命令を生成する方式を二つ紹介する．続いて，世代間互換性の問題を解決するための，仮想的な SIMD 命令を用いる動的 SIMD 命令変換を紹介する．

4-3-1 オペランド幅に着目した動的 SIMD 命令生成

多くの演算では，データパスのビット幅と比較して，そこで扱われるオペランドのビット幅が十分小さいことが知られている．例えば SPECint 95 ベンチマークでは整数演算は半数以上が 16 ビット以下のオペランドに対して行われている [1)]．この場合，例えば 32 ビットデータパスを考えると，そこで実行される演算の半分は上位 16 ビットを必要としない．この観察から，16 ビットオペランドに対する演算二つを同時に 32 ビットデータパスで実行しようとするのは，非常に自然な選択である．プログラムの実行時にならなければビット幅が明らかにならないオペランドを扱う演算に対して，それらのビット幅が十分小さいことを動的に検出し，それら複数の演算を一つの SIMD 命令にパッキングする方式が動的 SIMD 命令生成である [1)]．

図 4･3 を用いて，32 ビット整数データパスの場合の動的 SIMD 命令生成を説明する．図の上部は命令ウインドウであり，演算の種類と演算に用いられる二つのソースオペランドを保持している．“Zero16?” は，そのエントリが 16 ビット以下のオペランドに対する演算であるか否かを表している．この例では，一つ目と三つ目の add 演算が 16 ビット以下のオペランドに対する演算である．図の下部は 32 ビット演算器を表している．命令ウインドウ中の二つの add 命令はいずれも 16 ビット以下のオペランドに対するものなので，32 ビットの演算器で同時に実行可能である．ただし，16 ビット目からの桁上げを禁止できる工夫が必要である．

ここでは 32 ビット整数データパス上で 16 ビット演算を 2 並列実行する SIMD 命令を生成する例を扱っているが，64 ビットデータパス上で 16 ビット演算を 4 並列実行する SIMD 命令の生成や，浮動小数点データパス上での SIMD 命令生成なども同様に実現可能である．

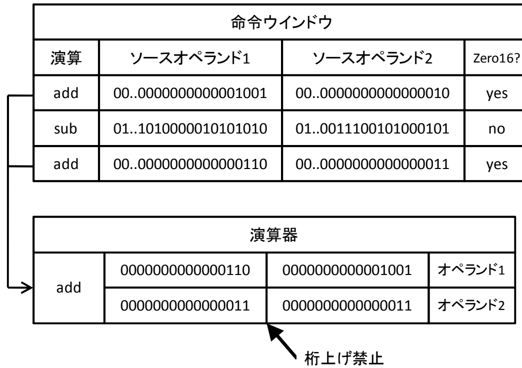

図 4・3　動的 SIMD 命令生成（文献 1）の Figure 17 をもとに作成）

4-3-2 ループ構造に着目した動的ベクトル命令生成

図 4･4 に示す投機的Dynamic Vectorization（DV）方式 [2)] [*1] ではロード命令を観察し，それが一定のストライドでメモリにアクセスしていることを検出すると，ベクトルロード命令に置き換えると同時にディスティネーションレジスタとしてベクトルレジスタを割り当てる．

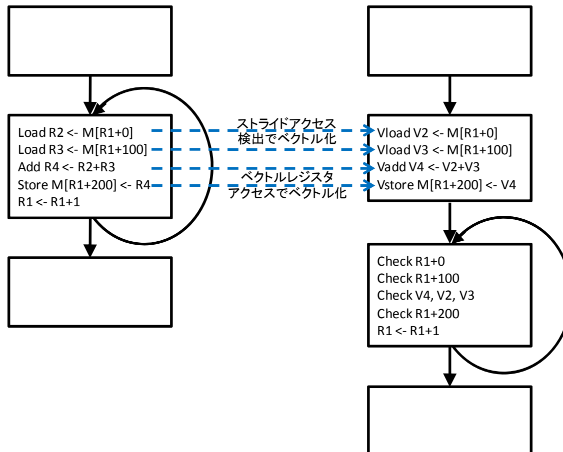

図 4・4　動的ベクトル命令生成

後続のスカラ命令は，もしソースオペランドがベクトルレジスタに割り当てられていると，ベクトル命令に置き換えられる．この時点ではベクトル化可能な命令であるかどうかは不明なので，その意味で投機的である．ループを思い浮かべてほしい．このスカラ命令はループ中に繰り返し出現するが，そのたびに実行するわけにはいかない．ベクトル化された時点で

[*1] 日本語にすると節題目と同じ動的ベクトル化方式となるので，混乱を避けるためにここでは DV 方式と呼ぶことにする．

実行されているからである．既にベクトル化されているスカラ命令は，このベクトル化が正しかったのか否かを検証する目的のみに利用される．レジスタマップ表を参照し，ベクトル化時にマップされたベクトルレジスタが依然としてソースレジスタに指定されているか否かを調べることで検証できる．

以上の説明では，投機的 DV 方式は古典的なベクトル命令を生成している．しかし，古典的なベクトル命令と SIMD 命令との間には本質的な違いはなく，ベクトルレジスタと SIMD レジスタとの違い，そしてベクトル演算の実現方式の違いに配慮すれば，投機的 DV 方式で SIMD 命令を生成可能であることがわかるだろう．

4-3-3 世代間互換性を考慮した動的 SIMD 命令変換

SIMD 並列性を利用するために多くの命令セットに SIMD 命令が追加されてきた．SIMD 命令は例えばコプロセッサとして実現されるアクセラレータ上で実行される．多くは 4～8 のデータ並列性を利用できるハードウェアが実装されているが，組込み応用での最適なベクトル長は 32 であるといわれており [3]，世代を経るにつれてハードウェアとして実装される SIMD 並列度は上がると予想されている．例えば，ARM の Neon SIMD 命令は 8 並列までの SIMD 演算が可能である．起源を同じにする命令セットであっても，世代ごとにハードウェア実装が異なるとバイナリ互換性が失われてしまう．この互換性の問題を解決するために，仮想的な SIMD 命令拡張である Liquid SIMD が提案されている [3]．

Liquid SIMD では, コンパイル時に生成された SIMD 命令を等価なスカラ命令に変換する．プログラム実行時にハードウェア支援によりスカラ命令をそのプラットフォーム上に実装されている SIMD アクセラレータに対応した SIMD 命令に再変換する．一度変換された SIMD 命令は専用キャッシュに保存されるので，以後の実行では変換することなく SIMD 命令を実行できる．また，バイナリ中には SIMD 命令は存在しないので，SIMD アクセラレータをもたないプラットフォーム上でも実行可能である．

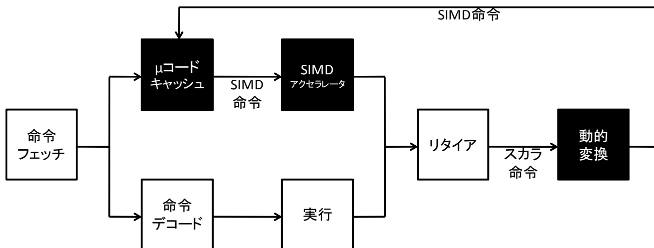

図 4・5　動的 SIMD **命令変換（文献** 3) **の** Figure 1 **をもとに作成）**

図 4･5 を用いて動的 SIMD 命令変換を説明する．まず，SIMD 化の対象となるスカラ命令がフェッチ, デコード, 実行, そしてリタイアされる. リタイア時に専用ハードウェアが SIMD 化の可能性をチェックする．SIMD 化可能なスカラ命令の集まりは関数として実現されており，対象となるスカラ命令の集まりを発見するのは容易である．SIMD 化可能な関数を発見できれば，SIMD 命令変換自体はコンパイル時に行ったスカラ命令変換の逆変換であり，状態遷移機械としてハードウェア実装することは容易である．動的変換された SIMD 命令は μ

コードキャッシュと呼ばれる専用キャッシュに保持される．これ以降，この関数が呼び出されるときは，μコード中の SIMD 命令に置き換えられて実行される．

Liquid SIMD ではコンパイラにより一度 SIMD 命令を生成する必要があるので，厳密な意味では動的 SIMD ベクトル化とはいえないが，現実的な規模のハードウェアにより互換性問題を解決できるという意味で重要性は高い．

■参考文献

1) D. Brooks and M. Martonosi, “Value-based clock gating and operation packing: dynamic strategies for improving processor power and performance,” ACM Transactions on Computer Systems, vol.18, no.2, pp.89-126, May 2000.
2) A. Pajuelo, A. Gonzalez, and M. Valero, “Speculative dynamic vectorization,” ACM SIGARCH Computer Architecture News, vol.30, no.2, pp.271-280, May 2002.
3) N. Clark, A. Hormati, S. Yehia, S. Mahlke, and K. Flautner, “Liquid SIMD: abstracting SIMD hardware using lightweight dynamic mapping”, Proceedings of 13th International Symposium on High Performance Computer Architecture, pp.216-227, Feb. 2007.

 © 電子情報通信学会 *2010*