■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）

5 章 専用コンピュータ

（執筆者：天野英晴）［2010 年 4 月 受領］

■概要■

専用コンピュータは一定の応用分野を指向して設計されているが，その得意分野を中心としてある程度の範囲に利用することが可能である．信号処理，画像処理，グラフィック処理，マルチメディア処理，ネットワーク処理などがその代表的な対象分野である．

汎用性，OS の必要性に縛られずに済むため，従来のコンピュータとは全く異なる構成を取ることが可能である．場合によっては，汎用 CPU に接続されて動作するアクセラレータ，あるいはオフロードエンジンの形を取ることがある．

専用コンピュータは，その応用分野およびアーキテクチャによって分類されるが，ここでは，きちんとした分類に従って紹介するよりも，よく用いられるキーワードを軸として，その周辺を解説する方法を用いた．

【本章の構成】

5-1 節は，古典的な専用処理プロセッサで今でも広く用いられる信号処理用プロセッサおよび，設計者が命令セットや構成をカスタマイズできるコンフィギャラブルプロセッサについて紹介する．5-2 節では，最近，性能向上と汎用化が目立つグラフィック用プロセッサを紹介する．5-3 節は，単一の命令で大量のデータを処理する SIMD 型アクセラレータ，画像処理用プロセッサについて紹介する．5-4 節は，メディア処理，画像処理等，一定の間隔で一定の大きさのデータを処理する必要のあるストリーム処理用プロセッサを紹介する．

■6 群 － 5 編 － 5 章

5-1 コンフィギャラブルプロセッサ

（執筆者：宮森　高）［2009 年 10 月 受領］

コンフィギャラブルプロセッサとは，メモリ構成や実装する命令などを変更でき，用途に適するようにカスタマイズすることができるプロセッサである．このカスタマイズの作業は RTL 設計時に行うので，実際に LSI ができた後でプロセッサの構成を変更することはできない．この点でリコンフィギャラブルとは異なっている．

5-1-1 背　景

2000 年を過ぎるころから，ディジタル携帯機器，ディジタル家電機器の機能・性能が著しく向上し，インターネットや携帯電話が急速に普及していった．これらのディジタル機器には，様々な機能を一つのチップに集積した SoC（System-on-a-Chip）である．SoC は特定の用途を想定して開発されるため, 個々のアプリケーションごとに要求性能･機能が異なること，核になるディジタル信号処理があることが特徴である．例えば，DVD プレーヤであれば MPEG-2 の復号化，ディジタル携帯オーディオプレーヤであればオーディオの復号化処理である．

また，2000 年の時点で先端であった 0.25 μm，0.18 μm プロセス以降では，組込み応用のマイクロプロセッサであれば，RTL から論理合成して実装しても性能や面積も十分実用になるようになってきた．このようは背景から，SoC に内蔵するプロセッサであれば，その応用に必要なディジタル信号処理を効率よく実現できるように，チップの設計にプロセッサの RTL を変更してカスタムプロセッサを開発すれば, より高い性能, より高い性能／面積効率, より高い性能／消費電力効率を実現することができる．

コンフィギャラブルプロセッサによって，このようなカスタムプロセッサを効率よく開発することができる．

5-1-2 コンフィギャラブルプロセッサのアーキテクチャ

コンフィギャラブルプロセッサは，シンプルなプロセッサコアをベースにして，予め用意された範囲でプロセッサの構成を変更したり，ユーザ独自の拡張機能を追加することができる．これによって，用途に特化したプロセッサを開発できる．コンフィギャラブルプロセッサの例としては, ARC International 社の ARC® Core [1], Tensilica 社の Xtensa® [2], 東芝の MeP [3], MIPS Technologies 社の Pro Series®ファミリーコアなどがある．

コンフィギャラブルプロセッサのプロセッサコア自体は，5 段から 7 段パイプラインのシングルスカラといったシンプルなものであり，スーパースカラやアウトオブオーダー実行などは取り入れていない．汎用的なプロセッサを目指しているものではなく，拡張機能によって特定用途の性能を引き出すことを想定しているため，プロセッサコア自身はなるべく小型で効率のよいものである必要があるからである．また，プロセッサコア自体の命令セットアーキテクチャは，予め決められており，小面積，低消費電力となるように最適に設計されている（プロセッサ記述言語で記述することで，プロセッサコアの命令セットも含めてカスタムプロセッサを開発するシステムも開発され，CoWare 社の Processor Designer やエイシッ

プ・ソリューションズ社の ASIP Meister として商用化されている．このようなプロセッサは ASIP（Application Specific Instruction Processor）と呼ばれている）．

コンフィギャラブルプロセッサは，以下のような構成を変更することができる．

・オプション命令の選択：乗算，除算，積和演算，飽和演算など
・メモリ構成の選択：キャッシュメモリやスクラッチパッド RAM の有無，サイズ，キャッシュの場合，連想度，ライン（ブロック）サイズなど
・機能の選択：例外処理，デバッグ機能

ARM や MIPS などの汎用的な組込み向けプロセッサでも論理合成可能なものは，キャッシュメモリやスクラッチパッド RAM の構成などを選択できるので，この点ではコンフィギャラブルプロセッサと変わらなくなってきている．

コンフィギャラブルプロセッサとしての，もう一つの特徴はプロセッサへのユーザ独自の命令やハードウェアエンジンを追加して，機能を拡張することができる点である．以下のような拡張方式から目標の性能，処理の内容などによって最適なものを選んだり，組み合わせて使うことができる．

・プロセッサコア命令の追加
・コプロセッサの追加
・ハードウェアエンジンの追加

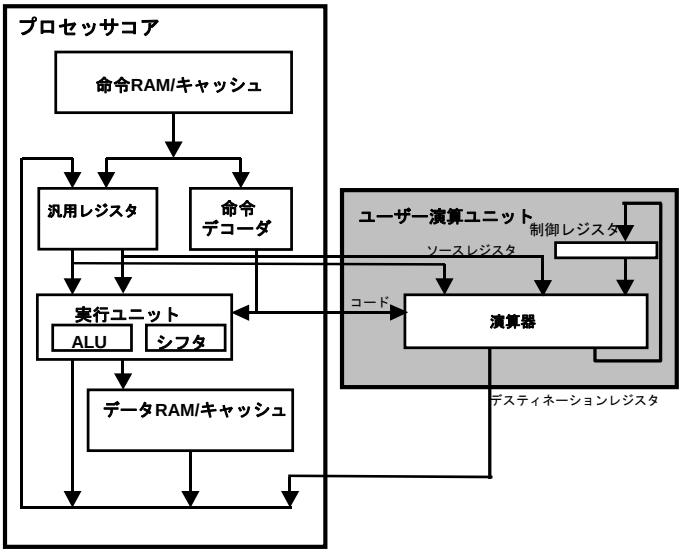

図 5・1　プロセッサコア命令の追加

プロセッサコアの命令追加は，ユーザ独自の拡張命令実行する演算ユニットを追加することで実現する．**図 5･1** のように，拡張演算ユニットは，ユーザ拡張命令のコードに従ってプロセッサコアの汎用レジスタから読み出した値や，拡張演算ユニット内のレジスタの値から，演算を行い，その結果をプロセッサコアの汎用レジスタへ書き戻すことが可能である．

コプロセッサ拡張は，プロセッサコアとは独立したデータパスをもたせることが可能な拡張である．コンフィギャラブルプロセッサだけでなく，一般的に浮動小数点ユニットや SIMD（Single Instruction Stream, Multiple Data Stream）型命令のコプロセッサなどがこの拡張方式で実装されている．コンフィギャラブルプロセッサによっては，プロセッサコアとコプロセッサで，同時に複数の命令を実行できる VLIW（Very Long Instruction Word）型のプロセッサを作れるなど，より複雑な拡張を設計できるものもある [3),4)]．

ハードウェアエンジンは，プロセッサコアとは独立にあるまとまった処理を行う拡張である．通常のプロセッサの場合は，プロセッサの外のバス上にこのようなハードウェア回路を置くことになるが，コンフィギャラブルプロセッサの場合は，専用のインタフェースでより高速にハードウェアエンジン内の制御レジスタをコントロールできるようにするなど工夫されている．

以上説明したとおり，コンフィギャラブルプロセッサでは，プロセッサの構成の選択と機能拡張によってカスタムプロセッサを設計できる．RTL などのハードウェア設計データだけでなくて，コンパイラやシミュレータが生成される．これによりアーキテクチャ設計時に，拡張機能を含めた性能評価を行うことができる．性能評価の結果，性能が不十分であれば拡張機能を見直せ，最適なデザインを探索できる．

5-1-3 コンフィギャラブルプロセッサを使った LSI 例

コンフィギャラブルプロセッサを用いて作られた SoC の例を示す．**図 5･2** は，MPEG-2 符号化・復号化（CODEC）用の LSI [5)] である．MPEG-2 に必要な処理をあげると以下の四つがある．

1. ビデオ CODEC 処理（圧縮と伸張の同時処理）
2. オーディオ CODEC 処理
3. ビデオとオーディオのビットストリームの分離と多重化
4. フィルタ，拡大・縮小，ウィンドウ処理などのビデオの前処理と後処理

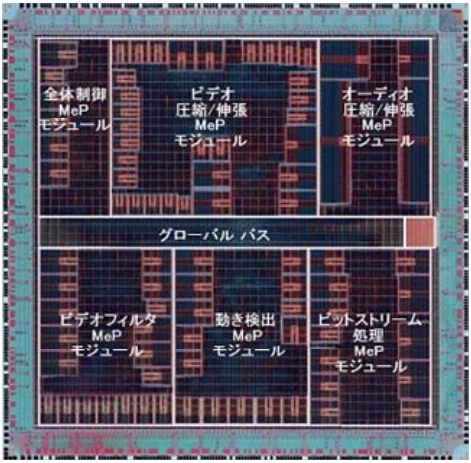

図 5･2　コンフィギャラブルプロセッサにより開発された MPEG-2 CODEC LSI

図 5･2 の CODEC LSI では，それぞれの処理をカスタマイズしたプロセッサ（MeP モジュール）が実行する．実際には，1. のビデオ CODEC のうち処理の重い「動き検出」を独立したプロセッサに分け，また，全体制御用に専用のプロセッサを設けたことで，合計六つのプロセッサから構成される．例えば，ビデオ圧縮・伸張プロセッサであれば，離散コサイン変換，量子化，動き保証，可変長符号化／復号化のためにハードウェアエンジンが拡張機能として追加されている．また，オーディオ符号化／復号化プロセッサでは，積和演算器をもつコプロセッサが追加され，汎用的な DSP（Digital Signal Processor）と同じようにソフトウェアでオーディオ処理を行うことができる．

■参考文献

1) 崔　一秀, “命令セットをユーザが設定できる CPU,” pp.59-66, Design Wave Magazine, 2000.
2) R. E. Gonzalez, “XTENSA: A Configurable and Extensible Processor,” pp.60-70, IEEE Micro, 2000.
3) T. Miyamori, “A Configurable and Extensible Media Processor,” Embedded Processor Forum, 2002.
4) A. Dixit, “Introducing The Xtensa LX Processor Generator,” Embedded Processor Forum, 2004.
5) S. Ishiwata, et al., “A Single Chip MPEG-2 Codec based on Customizable Media Microprocessor,” CICC 2002, pp.163-166, 2002.

■6 群 － 5 編 － 5 章

5-2 GPU

（執筆者：平澤将一）［2008 年 11 月 受領］

5-2-1 GPU の誕生

計算機においてビットマップ方式の GUI システムが発展しはじめた当初, CPU は現在と比較してはるかに低速であった．そのため，ビットマップシステムにおいて必要となる矩形データのコピーが CPU の重荷となっていた. CPU に代わって DMA などによりビットマップデータをコピーするグラフィックコントローラとして GPU（Graphics Processing Unit）は誕生した.

5-2-2 GPU の発展

1990 年代には計算機の性能向上に伴ってコンシューマ向けには Windows Operating System が，UNIX ワークステーション向けには CAD が普及していった．次第にグラフィック処理の統一的な API，ライブラリが必要となり，それぞれ Microsoft 社の DirectX，Silicon Graphics 社を中心とするグループによる OpenGL が策定，実装された.

ますます必要となるグラフィック処理能力と，統一された API によりソフトウェア API のハードウェア実装が進み，GPU の処理能力が増大していった.

GPU は，容易に交換して性能を向上させることが可能となるよう，主に汎用バススロットへ接続可能なグラフィックボードとして使用されていた. グラフィックボードには GPU のほか，GPU が使用するメインメモリ，メモリコントローラなどが実装されていた.

5-2-3 GPU の汎用化

1990 年代に普及した Windows 向け 2D アクセラレータとしての GPU の性能向上は次第に落ち着きつつあったが，2000 年代頃から 3D 処理を対象とすることで GPU の高機能化，高性能化は再びはじまった. コンシューマ向けには主に PC ゲームが GPU の性能を牽引していた.

3D 処理は，パイプライン化されて処理される．パイプラインには頂点演算，ジオメトリ処理，ピクセル処理などがある．初期の 3D 処理はソフトウェア処理されていたが，2D の場合と同様に GPU の実装に用いられるチップリソースが増加するに従いハードウェア実装され，高速化に貢献した.

しかし，PC ゲームを中心に 3D グラフィック処理に対する要求はますます複雑化し，ハードウェア実装されたパイプラインではニーズを満たせなくなっていった．そこで，パイプラインの処理をプログラム可能とするプログラマブルシェーダが用いられるようになった．頂点演算，ジオメトリ処理，ピクセル処理それぞれに対応するシェーダを設け，それぞれをプログラム可能とした．シェーダ向けのプログラム言語には，OpenGL 向けの GLSL，DirectX 向けの HLSL などが存在する.

各シェーダがプログラマビリティを増した結果，それぞれのシェーダ間の差異が減少していった．また，パイプライン処理において各ステージの処理量にばらつきがある場合，全体の処理性能が最も性能の低いステージにそろってしまう問題が顕在化した．この問題を解決するために，すべてのプログラマブルシェーダを同一アーキテクチャとし，各パイプライン

ステージの処理量に応じてシェーダの割り当てが可能となる統合シェーダが開発された．統合シェーダをもつ GPU においては, 各パイプラインステージの処理量にかかわらず高いシェーダ利用率を保つことが可能となった．

5-2-4 GPGPU

元来 GPU は極めてデータ並列度の高いグラフィック処理を専門に処理するプロセッサであり，キャッシュや分岐予測といった複雑な制御ロジックをもたない．限られたチップ面積をより多数のプロセッサに振り分けることが可能であり，高い演算性能をもっていた．プログラマブルシェーダにより高い演算性能をもつ GPU がプログラム制御可能となり, グラフィック処理以外の演算（高性能数値演算など）に用いる GPGPU（General Purpose computing on GPU）が注目され始めた．

初期の GPGPU においては OpenGL や DirectX といったグラフィック処理向け API を用い，それぞれに対応するシェーダ言語を用いてプログラムを行っていた．この場合，数値演算などユーザが意図するプログラム処理を，グラフィック処理パイプラインにマッピングする必要があった．

5-2-5 汎用プログラミング環境による GPGPU

ユーザが意図する処理をグラフィック処理パイプラインへマッピングすることなく GPGPU が可能となる汎用プログラミング環境が, NVIDIA 社の CUDA や, ATI 社の ATI Stream SDK を代表として登場してきている．これらのような汎用プログラミング環境を用いることで, C 言語に近い高級プログラミング言語を用いて GPU の演算処理をプログラム可能となり, シェーダ言語を用いたプログラムと比較して容易に GPGPU が可能となった．

CUDA と ATI Stream SDK を含めた GPU プログラミング環境には互換性がなく，ユーザは実行対象とする GPU ごとにプログラミングを行う必要がある．これに対して，GPU を含めて異なるアーキテクチャをもつ並列プロセッサで共通に用いることが可能となる仕様 OpenCL（Open Computing Language）が策定された．ただし 2008 年現在，OpenCL に対応した実装はなく，アーキテクチャごとに対応するプログラミング環境を用いることが必要な現状は変わっていない．

5-2-6 まとめと動向

高いデータ並列度をもつグラフィック処理を得意とする GPU は, 少数の命令を用いて多数の ALU により多数のデータを並列に演算するため非常に高いピーク性能をもつ．また，グラフィック処理においては単精度で十分な浮動小数点演算についても倍精度がサポートされつつあり，アクセラレータとして高性能コンピューティングに限らず，マルチメディア処理アプリケーションにおいてもますます利用されはじめている．

高いプログラマビリティによる汎用性をもった GPU 及び GPGPU はますます発展すると考えられるが，高いデータ並列性をもった実行モデルに対してどのようなプログラミングモデルを用いて活用するか，OpenCL の実装，活用を含めて学術界，産業界の対応が必要となっている．

■6 群 － 5 編 － 5 章

5-3 SIMD アクセラレータ，イメージプロセッサ

（執筆者：京　昭倫）［2009 年 6 月 受領］

プロセッサはその構成上，実際に演算を行う演算要素の部分（PE : Processing Element）と，命令やデータの供給や実行の流れを司る制御要素の部分（CE : Control Element）とに大きく分けられる．また一般に，PE側の設計 *1 が「ピーク性能」，CE側の設計 *2 が 1 以下の値をとる「PE稼働率」を定め，そして両者の積がプロセッサの実効性能を定める．しかしながら，ピーク性能がPEの規模にほぼ比例して向上されるのに対し，通常PEの数倍も実現コストがかかるCEは，その規模とPE稼働率との間には，明確な因果関係があるとは言い難い．実際，小規模のCEで大規模のPEを制御しても，高いPE稼働率を維持可能なケースが多く存在する．

こうしたことから，並列プロセッサを構成するに際しては，図 5･3 に示すように，より一般的な「CE と PE のペア」を並列処理の単位とする，いわゆる MIMD（Multiple Instruction Multiple Data）方式のみならず，CE の数を実質最小限の 1 個にまで減らすことで，同一のピーク性能を達成するのに大幅に実現コストを低減可能な，いわゆる SIMD（Single Instruction Multiple Data）方式が，古くから多くの並列プロセッサの構成方式として採用されてきた [1),2),5)].

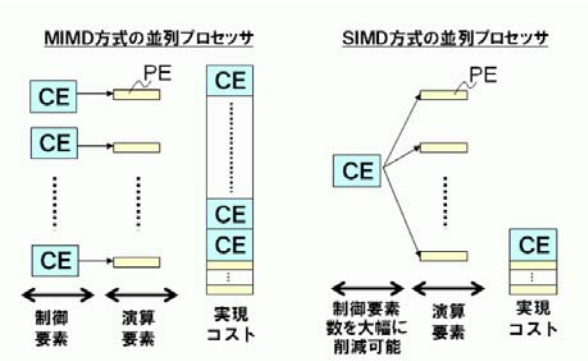

図 5･3　MIMD 方式と SIMD 方式の比較

このように SIMD 方式は MIMD 方式と比べると実現コストが大幅に低いため，同じダイ面積により多数の PE を集積できるようになり，ピーク性能向上に有効である．また，全 PE が同一制御下で動作するため，PE 間でのデータ交換性能の向上にも有効である．例えば，「SIMD 方式は MIMD 方式よりも，少なくとも 8 倍以上のピーク性能を実現できる」，「SIMD 方式では全 PE が同期して動作するため，PE 間でのデータ交換に要するオーバーヘッドは MIMD 方式の 1/200 程度で済む」といった報告 [4)] がある．実際，画像処理やグラフィックスといったメディア系の処理や，大規模科学技術計算などのように，大量のデータに対し同一の演算を施す，いわゆる「データ並列性」の顕著な処理に対しては，SIMD 方式のように単一命令流

*1　PE ごとデータパスの演算器類やレジスタ数，PE ごとの動作周波数，同時動作可能な演算器数，PE 数など．

*2　命令キャッシュ，分岐遅延制御，命令発行幅・Out-of-order 制御の有無，キャッシュ制御の粒度など．

でも，十分に高い PE 稼働率を実現できることが知られている．そこで，それらが主な処理対象の場合，SIMD 方式の採用は低コスト化・高性能化，あるいは優れたコスト性能比の達成の点で大きな効果が期待できる．

SIMD 方式に基づく並列プロセッサ（以降，SIMD プロセッサ）は，これまで「SIMD アクセラレータ」の名で，データ並列性が顕著なアプリケーションに対する CPU の性能不足を補う位置づけで，各種用途向けに開発されてきている．特に消費電力やダイ面積などの制約が厳しいうえに，高い性能も求められる組込み用途のイメージプロセッサ，画像処理プロセッサ，画像認識プロセッサは，多くが SIMD プロセッサに分類される [2), 6), 13), 14), 12)]．

以下では，まず SIMD プロセッサの基本構成について説明する．こうした基本構成に加え，更に対象アプリケーション分野ごとの演算ビット精度，要求性能，そして許容消費電力上限などの相違により，SIMD プロセッサの構成上でバリエーションが発生する．そこで，次に各分野での代表的な設計例を幾つか取り上げ，それらの概要構成を比較する．なお，SIMD 方式の効率性に着目し，CPU の命令セット内に SIMD 動作を指示する命令を追加する，いわゆる「SIMD 命令拡張」のアプローチ [7)] については，本編の他章で取り上げているため，本節では説明を割愛する．

5-3-1 基本構成

図 5･4 に示すように，一般に SIMD プロセッサは多数（数十～数千）の PE からなる PE アレイと，PE アレイに命令流を供給する一つの CE とで構成される．

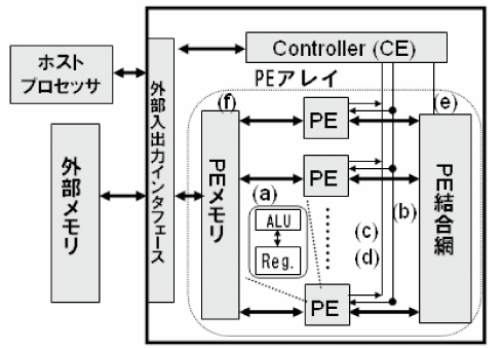

図 5・4　SIMD プロセッサの基本構成

CE は，PE アレイへ命令や共通するデータを放送して供給するとともに，通常のスカラープロセッサ相当の機能も併せもつ場合が多い．一方，PE アレイは通常，下記のような特徴をもつように設計されている．

・PE アレイ内の各 PE は，CE から放送される命令を受け取り，指定された演算器を一斉に動作させることで，PE 全体でベクタ演算を行う．PE ごとの演算器構成は，単純な ALU と少数のレジスタだけからなるものから（図 5･4(a)），多ウェイの VLIW 構成のものまで各種設計例がある．

・CE から PE アレイへは命令や共通するデータを放送するパス（図 5･4(b)）が存在し，それ

により命令や放送データは全 PE に同時に届けられる．本機能により，全 PE が共通に参照するデータは PE メモリ上ではなく，CE 内のデータメモリに一元的に保存することが可能となり，PE メモリの消費量を削減できる．

・全 PE からのステータス情報を統合し CE へ効率よく通知するデータパス（図 5･4(c)）が存在する．本パスは，PE 数分だけのデータを論理積あるいは論理和の形で圧縮して得られるスカラーデータ（PE 群の状態あるいは PE ステータス情報）を CE に送るのに利用される．PE ステータス情報は主に CE 上で，処理上での分岐の要否判断に使われる．

・特定 PE 内のデータを CE に効率よく読み出せるスカラーデータパス（図 5･4(d)）が存在する．CE が PE 番号を指定したうえで，図 5･4(d)のパスを利用することで，指定 PE 上のデータを読み出せるようになっている．

・PE 間のデータのやり取りに際しては PE 結合網（図 5･4(e)）が利用される．PE 結合網は，最も単純な一次元結合状（隣接する左右 PE との結合線のみが存在）のものから，全 PE がクロスバーで結合されているものまで，各種設計例が存在する．

・個々の PE への相異なるデータの供給には非常に多ビットの読出しポートか，多数の（例えば PE と同数程度の）バンクを備えたメモリ（図 5･4(f)）が利用される．

5-3-2 設計例

SIMD プロセッサは前項で説明した基本構成のもとで，更に対象とするアプリケーション分野の違いなどから，多くの設計バリエーションが存在する．**表 5･1** に，各主要分野ごとの，近年の代表的な SIMD プロセッサ設計例の諸元を示す．

表 5・1　SIMD プロセッサ設計例の概略諸元

プロセッサ名	GeForce8800	CSX700	Storm-1	CA1024	IMAPCAR2	Xetal2	Ri2006
想定利用形態	PC/サーバー			STB	組込み機器（車載、デジカメ他）		
アプリケーション分野	3Dグラフィックス	高性能計算	ビデオ・コーデック		画像認識	画像監視	画像処理
プロセス	90nm	90nm	130nm	130nm	90nm	90nm	90m
発表年度	2007	2008	2007	2006	2008	2007	2009
結合形態	N.A.	一次元	クロスバー	一次元	階層リング	一次元	一次元
PE数	8×16	96×2	16	1024	128	320	352
命令数/Cycle/PE	1	2	10	1	5	1	1
演算ビット数	32b浮動小数	64b浮動小数	32b整数	16b整数	16b整数	16b整数	16b整数
PEメモリ総量（KB）	256（16*16）	1152（576*2）	256	524	512	1280	352
PE毎レジスタ容量（B）	N.A.	128	64	16	58	2	32
消費電力	100W以上	10W以上	10W以上	5W以下	2W以下	1W以下	1W以下
動作周波数（MHz）	1350	250	800	200	108	84	200
ピーク性能（GOPS@16bit）	518GFLOPS	96 GFLOPS	256	200	138	107	70.4

3 次元グラフィックスがその本来の対象分野ではあるものの，科学技術計算分野などでの利用も広まりつつある GPU の一実現例である GeForce 8800 [10] は，その統合シェーダ部において，128 並列（8 並列×16 グループ）の SIMD 処理を 1 GHz 超の動作周波数で実行する．各 PE（文献 10）では SP : Streaming Processor）が同時に六つの実行主体（wrap）を維持でき，見掛け上，最大 768 並列で処理を行うことができる．プロセッサ内外間の動作周波数差が大

きいために発生する大きな外部メモリアクセス遅延を，wrap の実行切替えにより隠蔽している点に特徴がある．同様に科学技術計算を処理対象に想定し，各 PE に倍精度の浮動小数点演算器をもたせている CSX 700[11] は，96 PE のコアを二つ集積し 250 MHz で動作する．GeForce 8800 と比べて性能（動作周波数）は 1/5 程度にまで低下するものの，GeForce 8800 の 1/10 以下の消費電力を実現している．

コーデック用途を主に想定している Storm-1[9] は PE（文献 9）では lane）数が 16 と少なく，代わりに各 PE は五つの 32 bit ALU を有し，最大 80 並列で演算し 800 MHz で動作する．対照的に，同様にコーデック用途を想定している CA 1024[8] は 16 bit の ALU からなる単純な構成の PE を 1024 PE 集積し，200 MHz で動作させることで，要求性能の達成とともに消費電力の低減も実現している．

組込み用途向けに2 W以下程度の消費電力を実現する設計例としては, IMAPCAR 2, Xetal 2, そして Ri 2006 などがある. IMAPCAR 2[14] は 5 Way VLIW 構成の PE を 128 PE 集積し 108 MHz で動作し，また画像認識処理に一定の割合で存在するタスク並列性の活用を考慮し，MIMD 動作もサポート可能な動的 SIMD/MIMD 切替え型 SIMD プロセッサである．Ri 2006[12] はカメラ画像処理用途向けに，16 ビット ALU をもつ PE を 352 PE 集積し 200 MHz で動作する．同様に組込み画像処理用途を意識し，320 PE を集積した Xetal[13] は Ri 2006 と比べ，各 PE に更に 16 ビットの積和演算器を加えている代わりに, 動作周波数は Ri 2006 の約半分である 84 MHz に抑えられている．

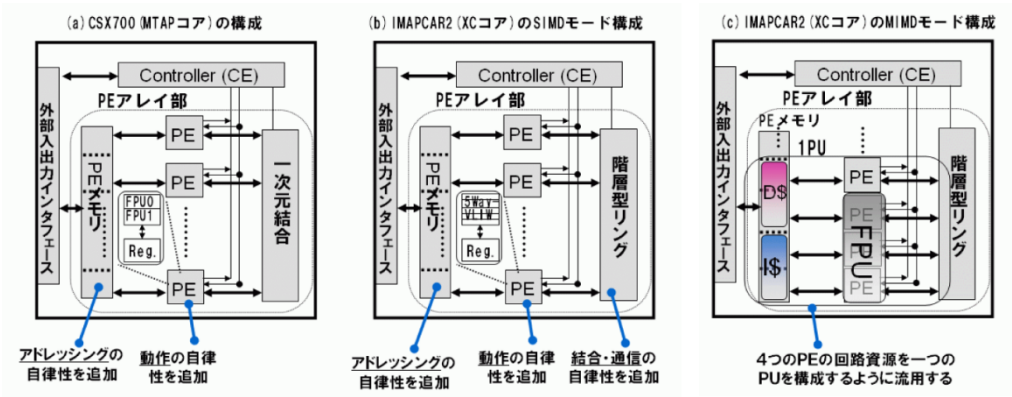

図 5・5　SIMD プロセッサの構成例

以下では，より詳細な SIMD プロセッサの設計アプローチの比較例として，CSX 700 と IMAPCAR 2 を取り上げる．**図 5･5**(a),(b)にそれぞれ，CSX 700 を構成するプロセッサコア MTAP，及び IMAPCAR 2 を構成するプロセッサコア XC の（SIMD モード時の）概要ブロック図を示す．両プロセッサはそのターゲットとするアプリケーション分野の違いにより，各 PE がもつ演算器の種類は大きく異なるが，PE ごとが動作・不動作を独立に決定できる点（動作の自律性の追加），PE ごとが相異なるメモリアドレスへアクセスできる点（アドレッシングの自律性の追加）など，各 PE への自律性付与[3] の方針に関しては類似したアプローチをとっている．一方で，XC コアでは更に，PE ごとが相異なる距離にある他 PE とデータ交換

できる機能（結合・通信の自律性）を追加し，かつ四つの PE を一つの独立動作可能な PU（Processing Unit）へ動的に構成を切り替える機能も備える．なお図 5･5(c) に示すように，四つの PE を単位に回路資源をできるだけ流用して PU を構成したことで, XC コアでの MIMD モードのサポートに伴う回路コスト増は 10 %程度に留まる [14].

5-3-3 今後の展望

SIMD プロセッサは，データ並列性が顕著な処理に対してはその構成上，非常に優れたコスト性能比を実現できることから，科学技術計算や画像などを扱うマルチメディア処理の高速化目的で近年，CPU のアクセラレータとしての導入が盛んである．また，これまでの研究開発により，VLIW 方式の導入による命令レベル並列性の活用や，複数 PE を束ねることによるタスクレベル並列性活用の可能性なども示されたことから，SIMD プロセッサの適用分野は今後ますます拡大していくこととなろう．一方で，SIMD プロセッサはこれまで設計ごとに，相異なるベクタタイプのデータ型拡張をもつ C 言語がプログラム開発で利用されてきている．そこで，今後は OpenCL に代表されるような，SIMD プロセッサに向けた C 言語拡張の共通化・標準化活動がますます重要な課題になると予想される．

■参考文献

1) W. D. Hillis, “The Connection Machine,” MIT Press, Cambridge, MA, 1985.
2) T. J. Fountain, K. N. Matthews, and M. J. B. Duff, “The CLIP7A Image Processor,” IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI), vol.10, no.3, pp.310-319, 1988.
3) T. J. Fountain, “Introducing local autonomy to processor arrays,” In: H.Freeman (ed.) Machine vision: algorithms, architectures and systems, Academic press, pp.31-56, 1988.
4) M. Wu and W. Shu, “MIMD programs on SIMD architectures,” Proc. of Sixth Symp. on the Frontiers of Massively Parallel Computing, pp.162-170, 1996.
5) D. W. Hammerstrom and D. P. Lulich, “Image Processing Using One- Dimensional Processor Arrays,” Proceedings of the IEEE, vol.84, no.7, pp.1005-1018, 1996.
6) M. Ishikawa, K. Ogawa, and T. Komuro, “A CMOS Vision Chip with SIMD Processing Element Array for 1ms Image Processing,” ISSCC Digest of Technical Papers, pp.206-207, 1999.
7) S. K. Raman, V. Pentkovski, and J. Keshava, “Implementing streaming SIMD extensions on the Pentium III processor,” IEEE Micro, vol.20, no.4, pp.47-57, 2000.
8) L. Bivolarski, et al., “The CA1024: A fully programmable system-on-chip for costeffective HDTV media processing,” Proc. of Hotchips 18, 2006.
9) B. Khailany, et al., “A Programmable 512 GOPS Stream Processor for Signal, Image, and Video Processing,” ISSCC Digest of Technical paper, 15.2, pp.272-273, 2007.
10) E. Lindholm and S. OBerman, “The NVIDIA GeForce 8800 GPU,” Proc. of Hotchips 19, 2007.
11) “CSX Processor Architecture,” http://www.clearspeed.com/
12) http://www.ricoh.co.jp/LSI/product assp/ri/
13) A. Abbo, et al., “XETAL-II: A 107 GOPS, 600mW Massively-Parallel Processor for Video Scene Analysis,” ISSCC Digest of Technical Papers, pp.270-271, 2007.
14) S. Kyo, et al., “IMAPCAR2: A Dynamic SIMD/MIMD Mode Switching Processor for Embedded Systems,” Proc. of Hotchips 21, 2009.

■6 群 － 5 編 － 5 章

5-4 ストリームプロセッサ

（執筆者：京　昭倫）［2009 年 8 月 受領］

「ストリームデータ」とは通常，大量の一連とする均質的なデータ列のことを指す．例えば，画像や音声，そしてグラフィックスなどの分野では，処理対象データがこうした「ストリームデータ」の形態をもつ．一方で，単体プロセッサの動作周波数向上が頭打ちするなか，今後も高まることが予想される画像，音声，グラフィックスといったメディア処理の性能要求を継続して満足していくためには，並列プロセッサの活用が必須となってきているなかで，近年，メディア処理向けに設計された並列プロセッサを「ストリームプロセッサ」と呼ぶ事例が増えてきている．しかしながら，これらの自称ストリームプロセッサと従来の，特に前節で取り上げた SIMD プロセッサとの間には，必ずしも明確な構成上の違いがあるわけではない．こうしたなかで本節では，ストリームプロセッサをストリームデータがもつ幾つかの特徴をうまく利用することで，同一の処理性能を汎用プロセッサよりもはるかに低いコストで実現できるように設計されたプロセッサの総称と捉えることとする．

以下ではまず，ストリームデータ処理に特化することで，汎用プロセッサの場合と比べ，ストリームプロセッサにはどのようなコスト性能比の改善に向けた設計上の選択肢が現れるかについて説明する．次に，ストリームプロセッサの典型的な実現例の構成を概観する．最後に今後の設計トレンドを説明する．

5-4-1 設計上の選択肢

ストリームプロセッサでは，ストリームデータがもつ「大量性」，「均質性」，そして「連続性」といった特徴をうまく利用することで，下記に示すように，汎用プロセッサと比べ，単位性能当たりの（ハードウェア）実現コストを下げられるようになるといえる．

・ストリームデータのもつ「大量性」そして「連続性」の特徴が，以下に示すように，メモリシステムの低コスト化設計，すなわち「スループット優先」，そして「キャッシュ機構の簡略化またはスクラッチパッド化」といった設計上の選択を可能にする．

1. ランダムアクセス性能よりもブロック転送性能にフォーカスした設計，すなわち遅延優先ではなく，より実現コストの低いスループット優先のメモリシステムの採用が可能となる．
2. ブロック単位でデータを扱えばよいことから，ハードウェアキャッシュ制御機構を装備する場合でも，管理情報のスリム化（キャッシュラインサイズを大きくしキャッシュタグの記憶領域を削減するなど），あるいはそもそもキャッシュ制御機構自体を省略しスクラッチパッド構成を採用することも選択肢の一つとなる．

・ストリームデータのもつ「大量性」そして「均質性」の特徴が，以下に示すように並列プロセッサ構成における低コスト化設計，すなわち「高並列化」，及び「SIMD 制御手法の採用」といった設計上の選択を可能にする．

1. ストリームデータがもつ「大量性」により，ある程度の長いストリーム長（ブロックサイズ）が想定できることから，高並列な演算器構成を採用しても高い稼働率を維持できる見込みが高い．そのため，性能目標を達成するのに動作周波数を上げるよりも，実現

コストの低い高並列構成の採用が可能となる．

2. ストリームデータがもつ「均質性」により，ストリーム内は通常同種のデータが並び，かつ各データに同一の処理を適用すればよい．そのため，並列化制御方式としてはより一般的な MIMD 方式ではなく，実現コストの低い SIMD 方式の採用が可能となる．

5-4-2 設計例

本項ではストリームプロセッサの典型的な実現例として，1990 年代より「ストリームプロセッサ」の名称で研究開発を進めてきているスタンフォード大学の Imagine [1] を紹介し，かつ前項で述べたストリームプロセッサがもつ特徴がどのように Imagine の設計で活かされているかについて言及する．

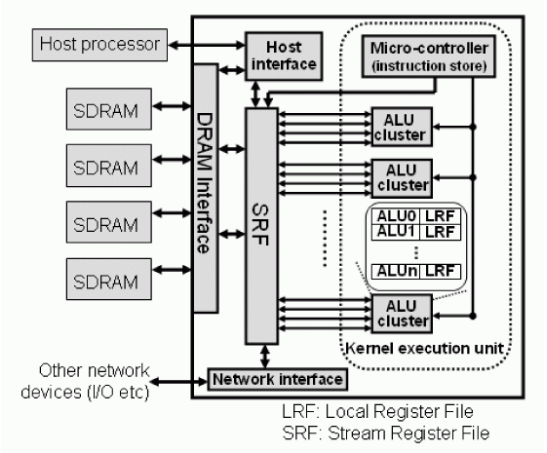

図 5・6　ストリームプロセッサ Imagine の構成 [3]

Imagine の構成を**図 5･6** に示す [3]．まず，演算は VLIW 実行をサポートするために複数の「ALU+LRF」で構成される ALU クラスタを多数配置して行われる（ストリームデータがもつ「大量性」の特徴を活用）．また，これらの ALU クラスタ群の制御には，一つの制御ユニットから送られてくる命令列に従って動作するという SIMD 型実行制御を採用している（ストリームデータがもつ「均質性」の特徴を活用）．これらにより，命令レベル並列性が ALU クラスタ内での VLIW 実行，そしてデータレベル並列性が多数の ALU クラスタによる並列処理により活用されるが，全体的には個々の PE（= ALU クラスタ）が VLIW 実行を行う SIMD プロセッサと同一の構成をもつといえる．類似構成をもつ SIMD プロセッサとして，例えば IMAP-CE [5]，IMAPCAR [6] などがあげられる．

メモリシステムに関しては，LRF（Local Register File），SRF（Stream Register File），そして外部メモリ（SDRAM）の 3 段階のメモリ階層をもち，チップ外の SDRAM，オンチップの SRF，そして ALU に直結した LRF の順で，バンド幅が広くなっており，ブロック単位で SDRAM 上のデータを内部の大容量オンチップメモリ（SRF）に転送したうえで，多数の ALU で一斉に処理を行うことを想定した，スループット重視の設計を採っている（ストリームデー

タがもつ「大量性」と「連続性」の特徴を活用）．また SRF に対応する階層のメモリは汎用プロセッサであれば，キャッシュ制御の構成を採るのに対し，Imagine の SRF はスクラッチパットメモリ構成である（ストリームデータがもつ「大量性」と「連続性」の特徴を活用）．これにより，実現コストが高いキャッシュ制御機構を採用した場合よりも，同一ダイ面積でより大容量の SRF を搭載できるようになる．また，ブロック単位のデータ処理が中心のため，キャッシュ制御の不在による性能やプログラマビリティへの影響も限定的と考えられる．なお，オンチップのスクラッチパッドメモリは SIMD プロセッサでは既に一般的であるが，近年では CELL [7] で採用され，また科学技術計算の分野でもその有効性が示されるようになってきている [8]．**表 5･2** に，実際に試作された Imagine プロセッサの諸元を参考に示す [4]．

表 5・2　Imagine プロセッサの諸元 [4]

動作周波数	< 288 MHz
ピーク性能	23 GOPS@16bit or 11.8 GFLOPS@32bit
ALU数	48（8 clusters, 6 ALU each）
演算ビット数	32bit（integer / floating point）
LRF合計サイズ（b/w）	9.6 Kbytes（1570 Gbps）
SRFサイズ（b/w）	128 Kbytes（92.2 Gbps）
SDRAM interface b/w	16.9 Gbps
Tr数	21 million
プロセス	0.15um TI process
電源電圧	1.5V
ダイ面積	2.6cm^2

5-4-3 今後の展望

ストリームプロセッサの名称を用いているかどうかはさておき，近年，性能向上が著しいグラフィックス処理用プロセッサ [9]，及び科学技術計算，画像圧縮伸張，画像処理，そして画像認識に向けた SIMD/MIMD プロセッサ，例えば CELL [7]，Imagine の製品化版である Storm-1 [2]，そして IMAPCAR 2 [10] などが示すように，メディア情報がもつストリームデータとしての特性をうまく利用すれば，汎用プロセッサよりもかなり優れたコスト性能比を実現できるようになる．こうしたことから，今後ともこの分野での研究開発が活発に進められると期待される．一方で，前節でも述べたように，現状では設計ごとに相異なる C 言語の拡張仕様が存在する状況にあり，プログラミング環境が必ずしも整備されているとは言い難い．プログラミング言語の共通化・標準化活動は，ストリームプロセッサの普及にとっても，今後ますます重要な課題になると予想される．

■参考文献

1) W. J. Dally, P.Hanrahan, and R.Fedkiw, “A Streaming Supercomputer, white paper,” Computer Systems Laboratory, Stanford Univ., Stanford, Calif., 2001.
2) B. Khailany, T. Williams, J. Lin, E. P. Long.; M. Rygh, D. W. Tovey, W. J. Dally, “A Programmable 512 GOPS Stream Processor for Signal, Image, and Video Processing,” IEEE Journal of Solid-State Circuits, vol.43, Issue 1, pp.202-213, 2008.
3) S. Rixner, W. J. Dally, U. J. Kapasi, B. Khailany, A. Lopez-Lagunas, P. Mattson and J. D. Owens, “Bandwidth-

Efficient Architecture for Media Processing,” Proc. of Int. Symp. on Microarchitecture (MICRO), pp.3-13, 1998.

4) U. J. Kapasi, S. Rixner, W. J. Dally, B. Khailany, Jung Ho Ahn, P. Mattson and J. D. Owens, “Programmable Stream Processors,” Computer, vol.36, Issue 8, pp.54-62, 2003.
5) S. Kyo, S. Okazaki, and T. Arai, “An Integrated Memory Array Processor Architecture for Embedded Image Recognition Systems,” 32nd Annual Int. Symp. on Computer Architecture (ISCA), pp.132-145, 2005.
6) S. Kyo, S. Okazaki, T. Koga, and F. Hidano, “A 100 GOPS In-vehicle Vision Processor for Pre-crash Safety Systems Based on a Ring Connected 128 4-Way VLIW Processing Elements,” Digest of Technical Papers of Symp. on VLSI Circuits, pp.28-29, 2008.
7) M. Gschwind, P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazaki, “A novel SIMD architecture for the Cell heterogeneous chip-multiprocessor,” Proc. of HOT CHIPS 17, 2005.
8) 中村　宏, 近藤正章, 大河原英喜, 朴　泰祐, “ハイパフォーマンスコンピューティング向けアーキテクチャ SCIMA,” 情報処理学会論文誌ハイパフォーマンスコンピューティングシステム, vol.41, no.5, pp.15-27, 2000.
9) E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, “NVIDIA Tesla: A Unified Graphics and Computing Architecture,” IEEE Micro, vo.28, Issue 2, pp.39-55, 2008.
10) S. Kyo, S. Nomoto, T. Koga, H. Lieske, and S. Okazaki, “IMAPCAR2: A Dynamic SIMD/MIMD Mode Switching Processor for Embedded Systems,” Proc. of HOT CHIPS 21, 2009.