新しいプロセッサアーキテクチャと アルチメディア／分散制御への応用

山崎　信行

■ 研究のねらい

本研究では、OAやFAの制御、インテリジェントビルディングの制御、超分散センサの制御、各種ロボットの制御、マルチメディア処理等の様々な並列分散リアルタイム処理を実現可能にする、リアルタイム通信ネットワークとリアルタイム処理を融合したアーキテクチャの研究を行なう。

現在、高速通信インタフェースとしてはATM, Fibre Channel, IEEE-1394, Gbit Ethernet等の規格が米国主導で提案・設計されている。これらの規格では、マルチメディアデータ用のリアルタイム通信のためのモード（アイソクロナス転送等）をサポートしている規格もあるが、リアルタイム通信モードではエラー訂正を行なっていないとかソフトリアルタイムしか実現していない等、並列分散制御に使用するためには多くの問題を抱えている。

一方、現在のマイクロプロセッサはできるだけピーク性能を引き出すためにスーパーパイプライン、スーパースカラー、アウトオブオーダ実行、VLIW等の要素技術を用いてより並列度が高く複雑な方向に同様に米国主導で進化を続けている。しかしながら、これらのプロセッサでは、高速演算性能を引き出すための上記のアーキテクチャのために、逆に実際の実行サイクル数が予測できなかったり、割り込みのペナルティが大き過ぎたりするために、リアルタイム制御には不向きなものとなっている。また、マルチメディア演算は大量のデータに対して同様の演算を繰り返し行うという性質があるが、そのようなマルチメディア演算を高速かつリアルタイムに行いたいという要求も同時に実現したい。

本研究では、従来別々に研究されていたネットワークのアーキテクチャとプロセッサのアーキテクチャを並列分散リアルタイム処理という観点から融合し、リアルタイム通信及びリアルタイム処理機能の両方を有した並列分散制御用リアルタイムプロセッサのアーキテクチャの研究を行なう。そしてそのアーキテクチャを元にリアルタイムネットワークとプロセッサを融合したReal-time Networking and Processing Unit (RNPU) の設計・実装を行なった。

■ 研究成果

1　リアルタイム通信：*Responsive Link*

レスポンシブリンクのリアルタイム通信アーキテクチャは、組み込み制御用途であることを十分に考慮して設計されている。一般的に通信データは通信の最小単位（ここではパケットと呼

ぶ）に分割されて送受信される。マルチタスクシステムでは同時に他の複数のシステムとデータをやり取りすることが要求されるので、それらのシステムでは時分割で並行処理を行い、複数の他のシステムとパケットのやり取りを行う。ここで、通常のデータのサイズ（センサデータ、画像データ、音声データ等）は大きく、それに対してイベントのサイズは非常に小さい。従ってこの方法では同時に通信すべき通信データとして、大量のデータパケットと、ごくわずかではあるがリアルタイム制御用途には非常に重要なイベントパケットが同一種類のパケットとして存在する。制御等を前提にするリアルタイムシステムにおいては、イベント伝達のレイテンシをできるだけ短くかつ固定にすること及び時間制約の厳守が最も重要であると考えられる。しかし、データとイベントを共有された同一の通信線を通して時分割に通信を行う従来方式ではイベント伝達の時間が正確にバウンドできないので、ハードリアルタイムシステムは実現困難であると考えられる。

また、複数のモジュールでひとつの通信チャネルを共有するシリアルバスでは、同時に何台のモジュールが通信するかによってバンド幅が動的に変化し時間をバウンドすることが困難であり、実効速度も出にくい。

さらに、リアルタイム通信におけるトレードオフとして、ソフトリアルタイム通信は主にマルチメディアデータの通信等に用いられ、ハードリアルタイム通信は主に制御等に用いられるので、

- ソフトリアルタイム：バンド幅保証 ⇒
  スループットをできるだけ上げたい
- ハードリアルタイム：レイテンシ保証 ⇒
  レイテンシをできるだけ小さくしたい

という要求がある。しかしながら表1に示すように、パケットサイズを大きくするとスループットは大きくなるが、同時にレイテンシも大きくなってしまう。逆にパケットサイズを小さくするとレイテンシは小さくなるが、オーバヘッドが大きくなりスループットが小さくなってしまう。

表 1: パケットサイズのトレードオフ

項目 \ パケットサイズ	大	小
スループット	大	小
レイテンシ	大	小

従ってデータラインとイベントラインを分離し、かつ各リンクの結合形態は point-to-point（1 対 1）で設計・実装する方針を取る（図 1 参照）。そして、データラインではパケットサイズを固定長で且つ大きめにしてバンド幅を保証するソフトリアルタイムを実現し、イベントラインではパケットサイズを固定長で且つ小さめにしてイベントの遅延時間を保証するハードリアルタイムを実現する。

データラインは通常のデータやソフトリアルタイム通信用データを通信するために用い、イベントラインはイベントを通信する目的にのみ使用する。イベントの大きさ及び流量は非常に小さく、データパスとイベントパスが分離されており、さらに point-to-point 通信を採用して

いるので、イベントパケットはイベントラインを通してハードリアルタイムに伝達することができる。(データラインがどんなに込み合っていたとしても関係ない。)

レスポンシブリンクのネットワークスイッチ部は追い越し機能を有している。パケットは固定長(データパケット:64byte、イベントパケット:16バイト)で、パケットに優先度が付加されている。データケットはアドレス(ソースとデスティネーション)、ペイロード、ステータスから構成される(図2参照)。衝突が起きない限りデータはヘッダ部受信のオーバヘッドのみで次ノードに転送されるが、あるノードで衝突が起こった場合は、優先度の高いパケットが低いパケットを追い越すことができるようになっている。

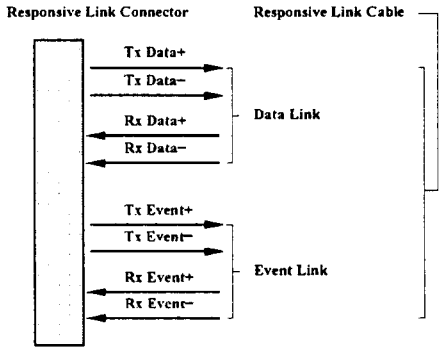

図 1: レスポンシブリンク

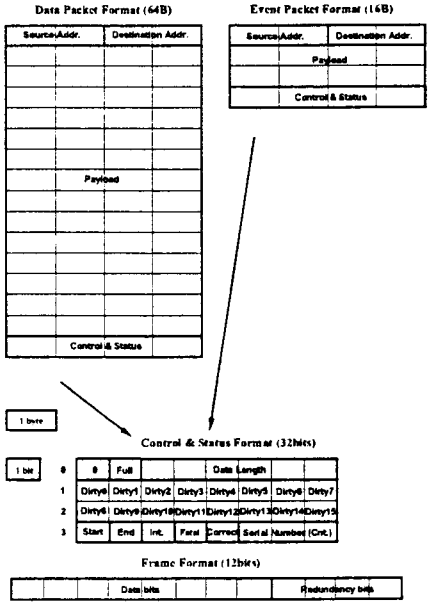

図 2: パケットフォーマット

また、パケットの優先度は、ノード毎に付けかえることが可能である。パケットの追い越しは、そのパケットに付加されていた優先度によって行われるが、自ノードから次ノードに送信されるときに新優先度に付けかえることができる。新優先度は、次ノードでの追い越しに使用される。

パケットの優先度付加や優先度付け代えはソフトウェア(並列分散リアルタイムオペレーティングシステム)によって行われる。これらの機能によって従来までの集中管理型ではなく分散管理型のリアルタイム通信を実現している。

データとイベントに関して独立にルーティングテーブルを用意し、それぞれ独立にルーティングを行えるようにしている。

レスポンシブリンクでは1ホップごとに前方エラー訂正(FEC)を行い、1frame(8bitデータ+4bit冗長符号)につき1bitのエラーであれば、再送することなしにハードウェアで誤り訂正を行うことができる(図2参照)。

通信速度は、様々な環境(コンフィギュレーション、アプリケーション)を想定し可変とする。レスポンシブプロセッサは組み込み用途を想定しているので、消費電力が大きな問題となる。一般に通信速度(動作周波数)を速くすれば消費電力が大きくなり、遅くすれば小さくなる。

組み込みでバッテリ駆動で使用し、通信速度はあまり必要ないが、消費電力をできるだけ抑えたいという要求があるかもしれないし、消費電力は気にしないからできるだけ性能をひき出

したいという要求もあるかもしれない。あるいは、非常にノイズの大きいところ（例えばシールドされていない大型AC モータの近傍等）で使用する場合には、速度を速くすると伝送路にノイズが載り過ぎてエラー訂正が間に合わない状況も考えられる。

表 2: *Responsive Link*における速度と消費電力の関係

Speed of modulation(Mbaud)	100	50	25	12.5	6.25
Speed of Data(Mbps)	67	33	17	8	4
Latency of Event(μsec)	3.1	6.2	12.5	25.0	50.5
Power (W)	0.2	0.1	0.05	0.02	0.01

現在のレスポンシブリンクの最大変調速度は 100[Mbaud] である。表 2 は、通信速度と消費電力の関係を示している。通信速度は動作中に動的に変更可能である。変調速度が 100[Mbaud] の際のイベント 1 パケット（ペイロード：8byte）のレイテンシは、

$$Latency\ of\ Event = 2[\mu sec] + 1[\mu sec] \times n[hops]$$

で表すことができる。送受信のオーバヘッドが 2[μsec] かかり、1[hop] 当り 1[μsec] かかる。

電気的なインタフェースには P-CML(Pseudo Current Mode Logic) を用い、基本的には差動型で入出力を行う。P-CML は外付の簡単な回路によって、LVTTL(Low Voltage TTL) 及び ECL(Emitter Coupled Logic) とインタフェース可能であり、基板上や短い距離を接続するには LVTTL を用いて直結し、長い距離を接続するには ECL + 光ファイバでインタフェースを行う。

2 リアルタイム演算

今日のマイクロプロセッサは、パイプラインとスーパースカラにより Instruction-level での並列性 (ILP) を利用して、パフォーマンスを向上している。しかし、最近の研究ではスーパースカラマイクロプロセッサでさえ、プロセッサの利用効率の限界に近づいている。

パフォーマンス向上のための解決法は、大きくふたつあると考えられる。2 つ以上の完全なプロセッサを 1 チップにまとめるオンチップマルチによるアプローチと、複数のスレッドの同時実行を実現するマルチスレッドによるアプローチがある。どちらも、中〜粗粒度の並列性を高めることができる。

マルチスレッドプロセッサは、複数のレジスタセットに各々異なるスレッドコンテキストを格納する。高速なコンテキストスイッチを実現できるので、実時間処理向きと考えられる。また、組み込み用途ではコスト（ダイサイズ、消費電力など）が問題視されるが、同じ並列度のオンチップマルチに比べるとマルチスレッドプロセッサの方がコストが小さい。そこで、本研究では、いくつかのスレッドから同時に命令を発行することができる Simultaneous multithreaded (SMT) processor 型の実時間プロセッシングコアの応用を考える。

この際、マルチメディア処理にはリアルタイム性が要求されるため、大量のデータに対し効率よく演算を行なわなければならない。それらの演算の多くは積和演算であるため、積和演算をハードウェアでサポートすることにより、高速な処理が期待される。そのため、専用の積和演算器を持つ DSP や、データの並列性に注目した SIMD 演算機能を持つプロセッサが開発され

てきた。しかし、DSP やMMX 等のSIMD 演算機能は積和演算を高速に実行するが、汎用演算（積積、和和、和積演算等）の高速処理はできない。

そこで、本研究では積和演算だけでなく、積積、和和、和積演算等の組合わせ演算も汎用的に柔軟かつ高速に処理する機構として、実行ユニットを連結するPipeline Chainingを提案・設計する。

以下に、設計を行った実時間プロセッシングコアのマルチスレッド部のアーキテクチャ概要を示す。

- 複数本のスレッド（レジスタセット）を有する。
- 複数命令同時イシューを実現する。
- 複数個の独立したパイプラインを有する。
- 各スレッドは別々のレジスタセットで実行する。スレッドと実行ユニット間で固定した割り当てはしない。
- スレッドの開始、停止、同期、I/O操作に関してイベント駆動可能なスレッド制御ユニットを有する。
- イシューステージでは全てのイシューバッファから同時に最大のイシューバンド幅まで命令を選択する。
- 複数の仮想ファンクショナルユニット(VFU)により構成される。
- スレッド毎に優先度を付加する。
- 各スレッドはVFUを共有するが、衝突があった場合は、優先度の高いスレッドがVFUを使用する。

上記のマジックナンバ（実際のスレッド数等）は、性能、コスト、プロセスルール等のトレードオフによって決定する。

プロセッシングコアに実時間処理をさせるために、コンテキストスイッチはハードウェアによって高速に行われる必要がある。また、スループットをあげるためには、各コンテキストはできるだけ並列実行を行うことが望ましい。同時に、本アーキテクチャは組み込み用途をターゲットにしているため、各種コスト（チップ面積、消費電力等）をできるだけ低く抑えたいという要求がある。

複数スレッドが独立に動作するためには、スレッドごとに完全なパイプラインを用意すればよい。そうすることによってパフォーマンスは向上するが、コストが大きくなるデメリットがある。そこで、本アーキテクチャでは、複数のパイプラインごとにプログラムカウンタを用意し独立動作を可能にし、さらにその各パイプラインが仮想ファンクショナルユニット(VFU)を共有することによりコストを下げ性能を向上させる試みを行う。

また、実時間処理を行うために、各スレッド（コンテキストセット）には優先度が付加されている。VFUは実際には複数のパイプラインによって共有されているため、競合が起こる可能性がある。競合が起こった際には、優先度の高いスレッドが優先的にそのVFUを使用する。

さらに、実時間処理を行うために各スレッドは対応するイベントに従い、開始、停止、同期を可能にする。これらはスレッド制御ユニットにより実現する。この際、イベントはレスポンシブリンク経由で外部プロセッサから実時間に発生・伝達することが可能となっている。

設計するRNPUは既存のコンパイラを利用可能にするため、SPARC命令セットを用いる。

RNPUは命令の種類により4段から8段のパイプラインステージを持つ。それを図4に示す。

このように命令の種類ごとにパイプラインステージの段数を変えることにより、無駄なステージを作らずに、処理効率を向上させることができる。

図3に示すように、実行ユニットの使用率を向上させるために、実行ユニット以降を4つのパイプラインで共有する。実行ユニットの数に対して、その実行ユニットを使用したいパイプラインの方が多ければ、優先度に従い実行ユニットを割当てる。

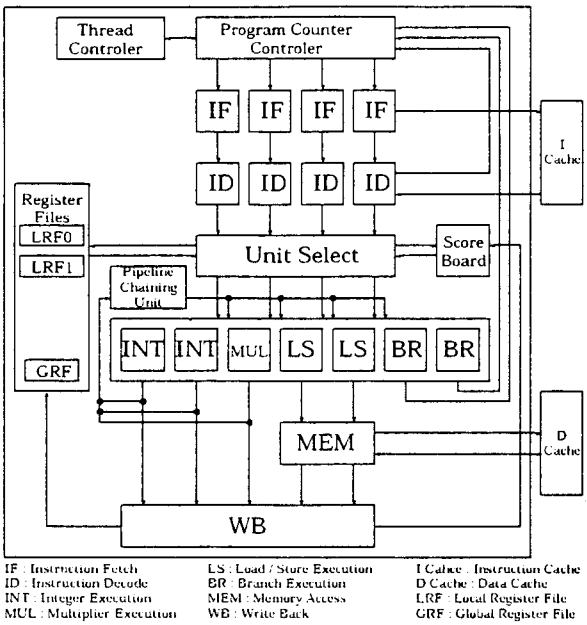

図 3: RNPUの構成

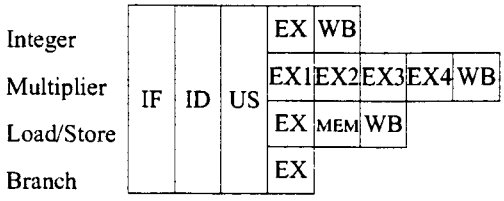

図 4: パイプラインステージ

Pipeline Chainingでは、積と和の実行ユニットを連結することにより、積和演算器にはない柔軟性の高い演算を実現する。また、RNPUでは、コンテキストスイッチを行ないながら、複数のスレッドが並列にパイプラインを流れている。そのため、以下のことが要求される。

1. 同時に複数のスレッドでPipeline Chainingを実行
2. 異なるスレッド間でPipeline Chainingを実行

これらの要求を満たすために、それぞれ実行ユニットを連結した時にデータを出す側と受け取る側の命令を対応付ける必要がある。そこで、識別子を用いて2つの命令間を対応付ける。

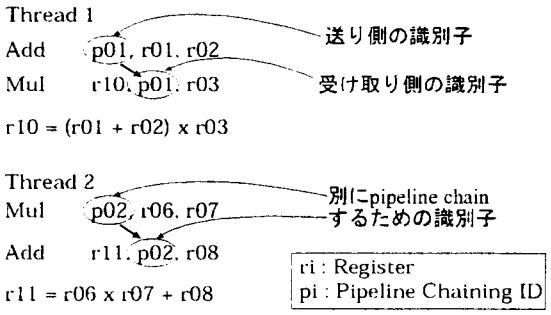

図 5: 識別子による対応付け

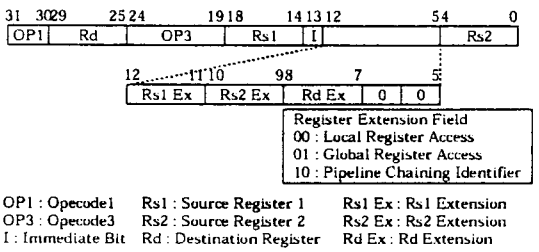

図 6: 命令フォーマット

図 5 に識別子を用いて対応付けを行った例を示す。例では Thread 1 と Thread 2 が並列に処理されていると仮定する。Thread 1 では ADD と MUL 命令を連結して和積演算を行なう。この場合 2 つの命令を対応付けるために p01 という識別子を用いている。同様に Thread 2 では MUL と ADD 命令を連結して積和演算を行なっている。2 つの命令を対応付ける識別子として p02 を使っている。

このようにして、複数のスレッドが同時に Pipeline Chaining をしようとした場合に適切な命令 (実行ユニット) 同士を連結する。

同様に、異なるスレッド同士の命令を連結する場合でも、送り側のスレッドと受け側のスレッドで同じ識別子を用いることにより、適切に対応する命令を連結する。

RNPU の命令フォーマットを図 6 に示す。SPARC 命令セットでは ADD、MUL などの論理演算、算術演算は即値を指定しない場合、12 ビットから 5 ビットは 0 にアサインされる。そこで、このフィールドを図のように 4 つに分割し、そのうち 3 つのフィールドを Rs1、Rs2、Rd フィールドがそれぞれレジスタを示すのか Pipeline Chaining のための識別子を示すのかを区別するために用いる。

本研究で設計する Pipeline Chaining のブロック図を図 7 に示す。

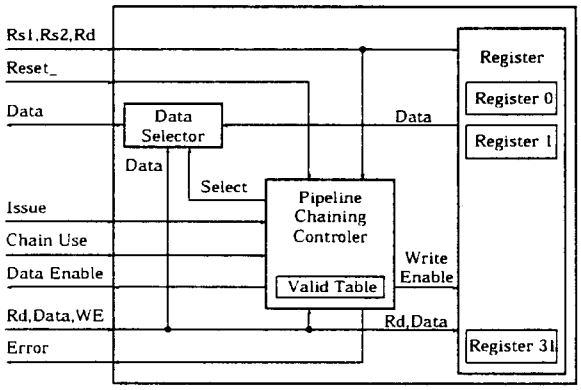

図 7: Pipeline Chaining のブロック図

Pipeline Chaining Unit は次の 3 つの構成要素から成る。

- 32 本のレジスタ
- Pipeline Chaining 制御ユニット
- データセレクタ

32 本のレジスタは受け取り側の命令がまだ来ていない場合に、値を保持するために用いる。

データセレクタは制御ユニットの信号により、実行ユニットから直接来たデータを受け取り側の実行ユニットに伝えるか、レジスタに保持されている値を受け取り側の実行ユニットに伝えるかを選択する。

制御ユニットは、それぞれどの実行ユニット同士を連結するかを管理する。受け取り側の命令がまだ来ていなければ、レジスタに値を保持し、Valid Bit を Enable にする。受け取り側の

命令に対しては、受け取るデータが用意できているかをData Enableで知らせる。データが用意できていなければ、その命令はストールし、適切なデータが到着するまで待つ。もし、レジスタから読み出す値が必要であれば、データセレクタに対してレジスタからの値を選択するよう要求する。また、Valid BitをDisableにして、値が受け取り側の命令に渡されたことを示す。データを送る側の命令に対しては、レジスタが適切な値を保持していて、受け取り側の命令がまだその値を使ってない場合はそれを知らせて、パイプラインをストールさせる。

これまで説明したリアルタイム通信リンク部（レスポンシブリンク）とリアルタイム演算部を融合して、RNPUを設計・実装した。具体的には、レスポンシブリンクのハードリアルタイムを保証するイベントリンクとリアルタイム演算部のイベント駆動制御部を密に接続し、1チップ化することによって、並列分散リアルタイム処理を実現するRNPUを実現した。

■ 今後の展開

現在、RNPUはシミュレーション及びFPGAを用いた評価ボードによって検証を行っている。FPGA評価ボードで論理検証と共にパフォーマンス（実時間性能、CPI等）に対する評価及びチューニングを行う。同時に、スレッド数、イシュー数、パイプライン数、リザベーションステーションのエントリ数等を変化させて実行・評価を行い、コストにみあった現実的な実時間プロセッシングコアをASICとして実現する。

このRNPUを用いたシステムでは、コントローラ間を通信リンクで接続するだけなので柔軟な構成が可能であり、ヒューマノイド型ロボットや台車型ロボット等の様々な形態のロボットの制御に応用できると考えられる。また、小型で接続方法が容易なので、簡単に壁などにも埋め込むことができ、オフィスオートメーションやホームオートメーション用のコントローラとして使用可能である。あるいは複雑化している車内制御システムであるとか、アミューズメント用途にも応用可能であると考えられる。

レスポンシブプロセッサのリアルタイム通信規格であるレスポンシブリンクはISO/IEC JTC1 SC25においてインタフェースとプロトコルの国際標準化を行っている。標準化することにより、機械システムや組み込みシステムの電子部を構築する際の共通プラットホームの実現を目指しており、性質の異なるシステムの相互接続を潜在的に可能にする。

■ 成果リスト

- 山崎 信行, 並列分散リアルタイム制御用レスポンシブプロセッサ, 日本ロボット学会誌, Vol.19, No.2, 2001. 掲載予定

プロセッサ・アーキテクチャの新展開

村上　和彰

マイクロプロセッサの応用分野が飛躍的に拡大（たとえば，従来のデスクトップ WS/PC から携帯情報端末，ネットワーク/通信機器といった高性能組込み機器へ）するにつれて，従来からの「高性能化」の要求に加えて「低消費電力化」といった新たな要求がプロセッサ・アーキテクチャに突き付けられている。これら２つの要件は図１に示すように互いに相反する要素を含んでおり，これらを同時に満足するためプロセッサ・アーキテクチャ上様々な技術開発が現在展開されている。以下，その概要を紹介する。

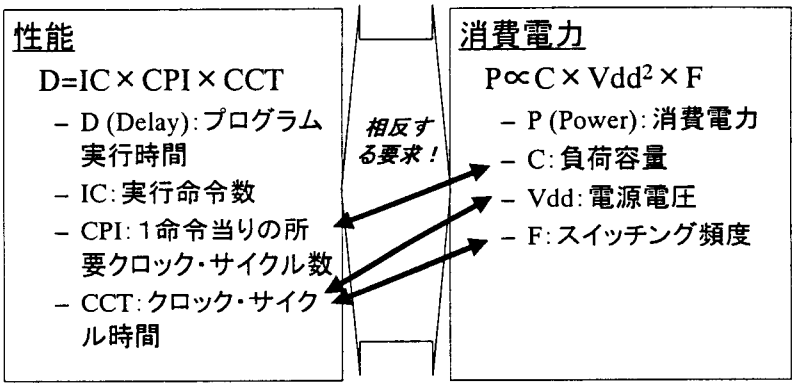

図１：性能と消費電力

●低消費電力性を維持しながらの高性能化技術

(1) VLIW（超長形式機械命令）アーキテクチャ

今日の高性能プロセッサのほとんどが OOO（アウト・オブ・オーダー）スーパースカラ方式をそのマイクロアーキテクチャとして採用している。本方式はオブジェクト・コードの互換性を維持しつつ，命令レベル並列性を活用して図１の CPI を低減しようというものである。しかしながら，命令レベル並列性を検出し活用するための回路が極めて大規模，複雑になってきており，その消費電力も無視できなくなってきている。そこで，同じく命令レベル並列性を活用するもののその処理の大部分はコンパイラに任せる VLIW 方式を採用したプロセッサが，近年多数登場している。VLIW 方式自身は別に新しい技術ではないが，オブジェクト・コードの互換性維持の困難さやコード・サイズの大きさが問題となって，今まではそれほど普及しなかった。が，ここにきて，Intel IA-64 の“Template”と“Bundle”というアイデア，Transmeta Crusoe の“Code Morphing”（動的バイナリコード変換）といった技術が誕生し，本格的な普及の段階を迎えつつある。

(2) “Make the Common Case Fast”

「プログラムの実行時間の 90%以上はプログラム中のわずか 10%の命令の実行によって費やされる」という“90/10 Locality Rule”が知られている。この性質を活用した代表的な技術に，キャッシュ・メモリや CISC の複合命令，等がある。最近では，さらに以下のような新しい技術が提案されている。

・“Make the Common Case Custom”（カスタム化）：かつての CISC の複合命令も本方式に分類される。「頻繁に実行される命令列に特化したハードウェアを設ける」というアプローチであり，近年では，

- Intel MMX に代表される SIMD 実行命令
- 一部の機能ユニットやコプロセッサの FPGA 化
- ユーザによる一部の命令セットのカスタム化

といった技術が実用化されている。

・"Make the Common Case Execute in Parallel"（部分的並列化）：VLIW 方式では，命令レベル並列度の高低に関わらず固定長の超長形式機械命令を用いるため，コード・サイズが大きくなるという問題がある。この問題を解決するために，「命令レベル並列度が低い部分はスカラ命令で，高い部分はスカラ命令を集めて VLIW 命令で」実行するハイパースカラ方式や iVLIW 方式といった技術が開発されている。

・"Make the Common Case Close to Processors"（メモリ・アクセス最適化）：「頻繁に実行される命令列をプロセッサに近いメモリに格納する」というアプローチである。命令キャッシュは本方式に近いが，必ずしも頻繁に実行される命令列のみを格納するわけではない。スクラッチパッド・メモリのようなプログラム制御可能な小容量メモリをキャッシュ・メモリとは別に設けて，命令フェッチの高速性と低消費電力性を同時に達成する。

(3) CMP（Chip MultiProcessor）と SMT（Simultaneous Multithreading）

令レベルよりも高位のスレッド・レベルの並列性を活用する。特に CMP は，ネットワーク/通信機器向けのネットワーク・プロセッサやコミュニケーション・プロセッサでの採用が顕著で，トランザクション・レベルの並列性を活用する。

(4) DRAM 混載

近年，プロセッサと DRAM それぞれの速度向上率の乖離が"Memory Wall"として問題視されている。DRAM 混載は主記憶デバイスである DRAM とプロセッサとを同一 LSI 上に搭載する技術であり，高いオンチップ・メモリ・バンド巾と低消費電力を同時に達成することを可能とする。グラフィックス・プロセッサとしての採用が盛んである。

●性能を維持しながらの低消費電力化技術："Divide and Reduce"

低消費電力化のためには図 1 に示した C，Vdd，F それぞれを低減する必要があるが，無秩序にこれらを低減すると性能まで低下させてしまう。そこで，性能を維持しつつ消費電力を削減するための手法として，図 2 に示すような"Divide and Reduce"技術が開発され一部実用化されている。

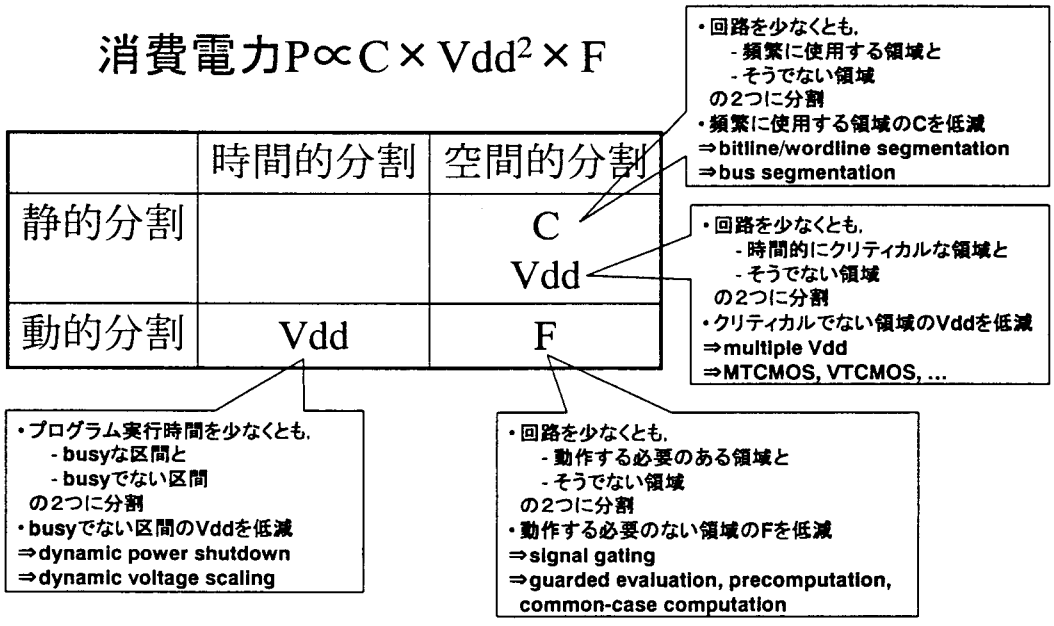

図 2 ："Divide and Reduce"