スーパーコンピュータの過去・現在・未来

内　田　啓一郎

Ⅰ．はじめに

2002年，地球シミュレータが稼動し，世界のスーパーコンピュータの地図が書き換えられた。1994年に，航空宇宙技術研究所のNWT（Numerical Wind Tunnelというスーパーコンピュータ）が世界一の座につき，しばらくその座を占めたあと，米国製スカラ並列コンピュータシステムが奪っていた世界一の座は，再び日本製ベクトル並列型システムによって取り戻された。

2002年11月，米国で開催されたSC2002（Super Computing 2002）では，地球シミュレータがLINPACK（線形計算用ライブラリ：標準試計算プログラム）性能及び実アプリケーション性能で，米国製に比べて５倍以上の性能を示し，ゴードンベル賞を受賞した。米国のスパコンユーザは，失地回復を目指して米国内でのスパコン設置要求を政府に突きつける絶好のチャンスといった形になっている。

並列システムという観点で言えばどちらも共通であるが，ピーク性能に対する実効性能の優劣という観点で，要素プロセッサをベクトルプロセッサ（ベクトル並列）とするかスカラプロセッサ（スカラ並列）とするかがひとつの議論である。

本解説記事では，できるだけ技術的な観点に注目した議論を行うつもりであり，ユーザからみた価値についても言及したい。始めにスーパーコンピュータの歴史的経緯を述べ，コンピュータアーキテクチャの進展について述べる。ベクトル処理技術も含めたプロセッサの内部構造について，並列処理技術の変遷を述べ，デバイス技術・設計技術の発展についても議論する。システムとしての並列処理技術について述べ，さらにコンパイラ技術に代表されるソフトウェアの発展を述べる。最後に，今後の見とおしを述べる。

ともすれば，スーパーコンピュータビジネスは政治に密着してしまう場面が多かった。つまり，スーパーコンピュータの処理能力が，国防能力に深く関係すると言うような国家政策（ナショナルセキュリティ）議論にすりかえられ，政治的観点からの議論が先行することがあった。したがって，スーパーコンピュータに関する議論が，純粋に技術的議論に留まらないことがあることに注意することが必要である。

しかしながら，政治が絡むことにより，国家威信からの国策スーパーコンピュータプロジェクトも成り立ち，大規模予算が取得できてきたということもある。一方，それがために純粋な商業利益ベースでの議論が成り立たない場合もある。したがって民間企業の事業としては，スーパーコンピュータ事業は微妙な面を持つと言わなければならない。

本記事はスーパーコンピュータ開発に従事していた者の立場から見た私見であり，ユーザから

の視点に乏しいかもしれないが，読者の参考になれば幸いである。

Ⅱ．スーパーコンピュータとは

スーパーコンピュータという言葉の定義は，その時代の一般的な計算機に比べて，特に優れた性能を持つコンピュータのことで，特別なコンピュータ構成方式を示すものではない。したがって，かつては最高速の汎用計算機システムをスーパーコンピュータと言ったこともあった。

しかしながら，1960年代から，スーパーコンピュータと言えば，主に科学技術計算用に使われる最高速のコンピュータを指すことが一般的であり，流体力学などに使われる偏微分方程式や構造解析用の有限要素法などを超高速で処理可能な科学技術計算に特化した計算機と理解されている。具体的にいうと，FORTRANで書かれたプログラム中でも一番処理時間がかかるDOループによる繰り返しを極力高速化することが，スーパーコンピュータの課題であった。

初期のスカラ型スーパーコンピュータには1960年代初めのIBM7030，LARC，1960年代終わりのIBM360-90，CDC6600/7600などがあった。DOループの高速化を実現するための工夫は古くから行われており，例えばIBM7030ではインデックスレジスタを採用した。これは，インデックスレジスタというDOループ処理専用のハードウェアを用意することで，ループ内での処理対象データのアドレス計算のハードウェア化を実現し，ループ処理のかなりの高速化を達成した。

DOループ処理は，ループ内の処理対象となっているデータを，一命令で一度に複数処理できれば著しく高速化できる。このような方式をSIMD（Single Instruction Stream Multiple Data Stream）処理という。この，SIMD型処理の方式には二とおりある。ひとつは，パイプライン演算器を使う方式であって，つぎ々とデータをパイプライン演算器に流しこんで，一個の命令（例えば加算）で，多数データの一括処理を実現したのがベクトル処理方式である。この様式を採用した初期のベクトルプロセッサの典型はSTAR-100である。SIMDのもうひとつの方式は，同じ命令をたくさんの並列計算機に放送し，異なるデータの処理を，たくさんの計算機で同時に実行することで高速計算を実現するものである。この方式の計算機が並列プロセッサ計算機であり，最初の大規模実現機はILLIAC　Ⅳであった。したがって，SIMDの代表としての二つの方式，つまりベクトルプロセッサと並列プロセッサの両方式は35年前にすでに出現していた。

商用のスーパーコンピュータは1970年代の初めから1980年代まで20年ほど，ベクトルプロセッサが主流の時代であった。ILLIAC　Ⅳ型の並列処理方式は，当時のコンパイラ技術のレベルではハードウェアの性能を生かせない上に，価格面でベクトル型に対抗できなかった。

この命令放送並列処理方式は，各プロセッサが独自の命令を実行するMIMD（Multiple Instruction Stream Multiple Data Stream）方式の実現によって，スカラプロセッサ並列型スーパーコンピュータシステムとして近年花開いた。

1990年代に入るとベクトルプロセッサ単体ではユーザの要求性能を満たすことができなくなった。この時期，ハードウェア価格も低下したため，ベクトル型でも複数プロセッサが実現され，ベクトル並列システムへと発展した。同時期のスカラ型の並列システムはMPP（Massively Parallel Processor）と呼ばれたが，ビジネスとしては生き残った企業はほとんどなく終わってし

まった。

現在のスーパーコンピュータは，並列処理システム方式（プロセッサレベルではMIMD方式）となっており，その構成要素としてはベクトルプロセッサ，スカラプロセッサの両者がある。ともに多数のプロセッサを並べて，プロセッサ相互間を結合ネットワークで結合している。このような並列システムを可能にした第一の理由は，ハードウェア価格の低減であり，プロセッサ数で，数百台から1000台を越える規模のシステムも可能になっている。さらに並列システム用OS及び言語処理システムの性能向上によって，スカラ型の大規模な並列処理システムの価格性能比が大幅に向上した。これらによって，スカラ並列型のスーパーコンピュータの商用化が実現された。

並列型のスーパーコンピュータの方式設計において，要素プロセッサとしてスカラ型を採用するかベクトル方式を採用するかは，商用システムでは価格性能比の観点から議論しなければならない。性能だけを考えれば，ベクトル型プロセッサのような高性能プロセッサを少数台並べるほうがプログラムの容易さやプログラムの実行効率の面から見て，良いに決まっているが，残念ながら，コストが高くなる。逆に，そこそこの性能のプロセッサを多数台並べると，価格は低めになるが，ユーザアプリケーションの実効性能は，プロセッサ間のネットワーク性能で決まってしまう。また，性能を引き出すためのプログラムの記述がベクトル型にくらべて格段に面倒になる。

しかし，実際のシステム性能の優劣は，単純に判定できず，ユーザの分野やアプリケーションプログラムによって大きく異なるため，システムの選択はユーザの使用環境によって大きく左右される。

Ⅲ．高速化と並列化（計算機アーキテクチャの歴史）

１．単一CPUでの高速化

フォンノイマン型のコンピュータの動作原理は，プログラム内蔵方式であり，記憶装置上の命令を逐次読み出して，逐次実行する。この形式のコンピュータでは，命令の処理プロセスは，命令を記憶装置から読出し解釈するプロセスと命令を実行するプロセスに分かれる。命令を実行するプロセスでは，演算によっては，記憶装置からのデータの読出しが必要になる。したがって，命令の実行には，最低１回，場合によっては２回の記憶装置へのアクセスが発生する。このアクセスの発生による負担は，記憶装置の応答がCPU内の処理に比べて非常に大きいために，計算機の高速化を妨げる要因のひとつにあげられている。これを，フォンノイマンボトルネックと呼ぶことがある。

フォンノイマン型の特徴は，この命令実行のプロセスを，逐次連続的に実行していくことである。計算機アーキテクチャの発展史は，フォンノイマンボトルネックの解消と命令の逐次処理の並列化（もしくは先行制御）技術の開発史でもある。

２．初期の並列化技術

ごく初期の計算機においては，図１のように，逐次型の加算器によって，レジスタＡとＢ上の数を１ビットずつ，語長32ビットまで繰り返し演算するため，32サイクルの時間がかかった。

図２のような並列式の加算器では，語長全部のビットを１サイクルで加算できるため，それだけで数10倍の性能向上が得られることになる。

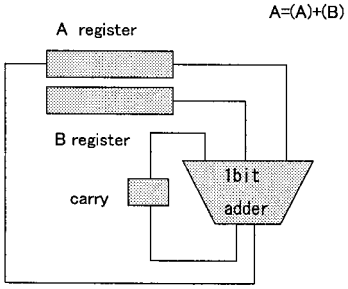


図１　逐次型加算器

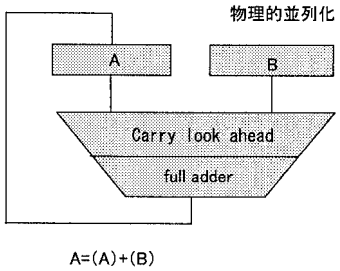


図２　並列型加算器

計算機のレジスタ方式は，レジスタがアキュムレータと呼ばれた１レジスタ方式から始まったが，現在の計算機の演算に利用できる汎用レジスタ数は64ないし128個に達しており，レジスタに保持できる情報量は100倍になっている。これに伴って，主記憶へのアクセス回数が低減され，演算性能が向上する。これもたくさんのレジスタを並列に設置するという意味での並列化である。

３．パイプライン制御

フォンノイマン計算機では，逐次処理が基本であるが，ハードウェアの追加で，現在処理中の命令に加えて，後に続く数命令をあらかじめ部分的に実行するとか，後続命令の実行準備をすることができると性能向上に効果が大きい。条件つきジャンプやプログラム中での後続命令の書き換えは先回り実行の効果を阻害する要因であるが，このような事態が発生する率は，それほど高くないので，命令の先行実行はCPUの高速化に非常に有効な方法である。

この命令先行実行を行う手段がパイプライン命令制御方式である。パイプライン命令制御方式では，命令処理の手順を数段から10段くらいのステージに分割して実行する。図３はパイプライン制御回路の例で，IDFESと言う５段ステージをもったパイプライン命令制御回路である。Iは命令読出し，Dは命令解釈，Fはデータ読み出し，Eは演算実行，Sは演算結果の書き込みを行うステージである。

パイプライン処理回路では，命令はパイプラインを通過していくうちに処理されるが，各ステージに独立の命令を入れることができるので，ある時間断面で見れば，同時に複数個の命令が並列に実行されていることになる。このように，パイプライン方式は，時間を細かく切って，各ステージを専門機能化し，ステージ間での流れ作業を実現することで，命令の並列実行を実現している。

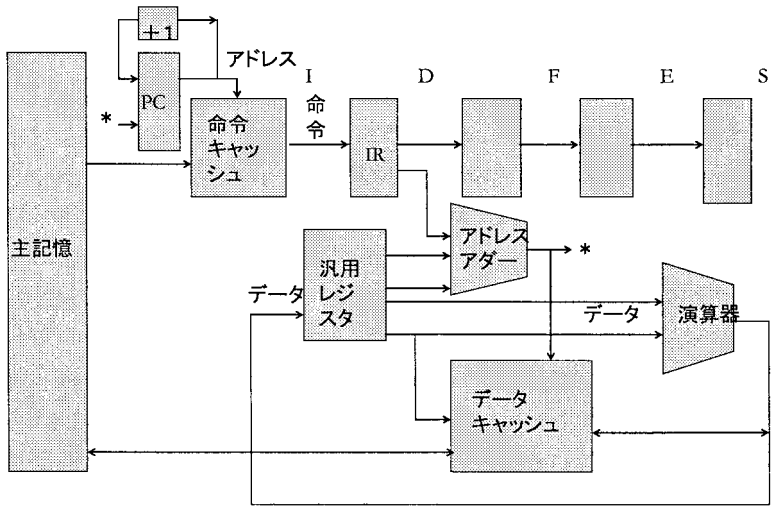


図３　命令制御回路

４．より高度な高速化技法

10年ほど前に，２命令を同一クロックサイクルの中で実行する命令レベル並列処理アーキテクチャ（ILP: Instruction Level Parallel Architecture）が生まれ，スカラプロセッサとしては画期的な性能向上をもたらした。この方式は，現在さらに複雑化しており，１サイクルタイムに６～８命令程度の複数命令を同時実行するものが出現している。この方式は，複数の命令パイプライン回路を設置して，同時に複数命令を実行する。このILP方式には，スーパースカラ方式とVLIW（Very Large Instruction Word）方式とがある。

ベクトルプロセッサでは，SIMD処理を実行するためのプロセッサとして設計された。すなわち，ひとつのベクトル命令（Single Instruction）をたくさんのデータ（Multiple Data）に対してベクトルパイプライン演算器で処理する。すなわちベクトルプロセッサはベクトルパイプライン演算器による，時間的並列化としての究極の姿である。

以上のように，計算機の性能向上の歴史は，単一処理から複数並列処理への計算機プロセッサアーキテクチャの進歩の歴史といえる。

５．システムとしての高速化

同様に，システムレベルでも，単一プロセッサから複数プロセッサ並列への流れをたどってきた。メモリ共有型のマルチプロセッサシステムは35年ほど前から実用化されている。現状ではメモリ共有型，分散メモリ型，その両方を併せ持つ方式の多数台の並列マルチプロセッサシステムが商用化されている。

科学技術計算の分野では，多数台のプロセッサを並列に使用して計算時間が短縮できる問題があり，そのようなアプリケーションの実行用途に，多数プロセッサ並列システムが導入されている。また，このようなシステムは，複数のプログラムを多数同時実行しても，価格性能比的に有

利であれば採用する理由となる。スーパーコンピュータにおいても，並列システムが主流となった。

ベクトルプロセッサでも，1994年のNWT以来（VPP-500として製品化），ベクトルパラレル（VPP）方式と呼ばれるプロセッサを多数並べる空間的並列システムの形態が一般的になった。

以上のように高速化の歴史はとりもなおさず，並列化の歴史であると言っても過言ではない。

Ⅳ．ベクトル型スーパーコンピュータ

以下では，ベクトルプロセッサの高速な理由を，命令処理及びメモリアクセスに着目し，説明を試みる。

１．ベクトルプロセッサの命令処理概要

図４にベクトルプロセッサの典型としてVP-200のシステム構成を示す。ベクトルプロセッサはベクトルレジスタと呼ばれる大容量のレジスタ群を持つ。VP-200のベクトルレジスタの総容量は64KB（キロバイト）で，これを256個のベクトルレジスタに分割可能である。１個のベクトルレジスタは32個の64ビットデータ（倍精度実数型）からなる。ベクトルレジスタ上での実際のベクトルデータの長さは，ベクトル長（倍精度実数型データの個数）レジスタで指定する。VP-200では，32個までのベクトルレジスタを連結してベクトル長1024のベクトルレジスタを構成できる。

VP-200は，さらに条件付演算用に同様な構成のマスクレジスタを持っている。マスクレジスタの個々のデータは真偽を示す１ビットの論理データである。

VP-200のベクトルユニットは，２本のロードストア，１本ずつの加算，乗算，除算，マスク演算用の合計６本のベクトルパイプライン演算器をもち，これらが並列に動作して，ベクトル演算を実行する。VP-200では，ベクトルユニットに加えて，システム全体の制御及び命令制御とスカラ演算を行うスカラユニットを用意している。

主記憶装置には２本のロードストアパイプライン，スカラユニットのキャッシュ，及び入出力チャネルからアクセスするためのバスがつながれている。

先に述べたようにベクトルプロセッサはDOループマシンである。しかし処理の仕方は単純にDOループをスカラプロセッサと同じように扱うのではない。下記のDOループを例にとって，スカラプロセッサとベクトルプロセッサの動作の違いを説明する。

```
DO   I=1,100
A(I)=B(I)+C(I)＊D(I)
END DO
```

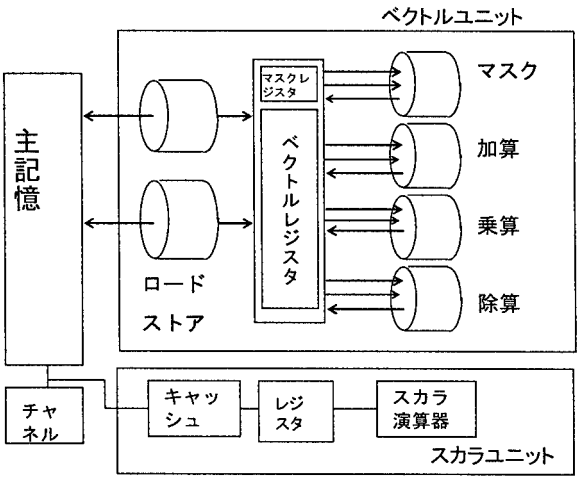


図４　VP-200のシステム構成

この例をスカラプロセッサで実行すると，以下のようになる。

① Iを１にセット
② C(I)のレジスタへのロード
③ D(I)のレジスタへのロード
④ C(I)とD(I)の積
⑤ B(I)のレジスタへのロード
⑥ B(I)と④の和
⑦ ⑥をA(I)にストア
⑧ I=I+1

このようにスカラプロセッサでは，同じインデックス値でDOループを一回実行し，インデックスを１だけ加算して，同じ処理を繰り返す。

一方ベクトルプロセッサではCのロードをI=１～100について実行し，Dのロードを Cのロードに続いてI=１～100について実行する。さらに，ベクトル乗算をI=１～100について実行し，Bのロード，加算，Aへのストアと連続して実行する。ここでのロード，加算及びストア命令は，すべてベクトル命令で，ベクトルロード，ベクトル加算，ベクトルストア命令である。ベクトル命令相互間では命令実行のオーバラップが可能なので，さらに実行時間を短縮することができる。このように，ベクトル処理ではベクトル命令単位でインデックスを１から最大値（ベクトル長：上記の例では100）まで繰返し，命令を完了させる。すなわち個別のベクトル命令ごとにDOループ回転を行う。

スカラプロセッサでは大きく回転するところをベクトルプロセッサでは個別ベクトル命令単位の小さな回転によって繰返し演算をして，それらの命令の連続によって，DOループを実行していく。つまりスカラプロセッサがループを縦割りに実行するとすれば，ベクトルプロセッサはルー

プを横割りに実行することになる。

２．ベクトルプロセッサの命令処理の詳細

より詳細に，先述のプログラムの実行を眺めてみる。

ベクトルロード命令（VLD）では，ベクトル長分のデータを繰返し処理によって主記憶からベクトルレジスタに持ってくる。ベクトルロード命令での主記憶へのロードアクセスは図５に示すようにパイプライン回路になっていて，アドレス計算，優先順位決定回路，メモリアクセスアドレス転送回路，データフェッチ回路，データの整合回路，ベクトルレジスタ（VR）への書込み回路などの回路を順次経由して，データを流れ作業的に伝送する。１から100までのデータが各段を流れていき同時に６個のデータが処理されている。これがベクトルロードパイプラインの動作原理である。

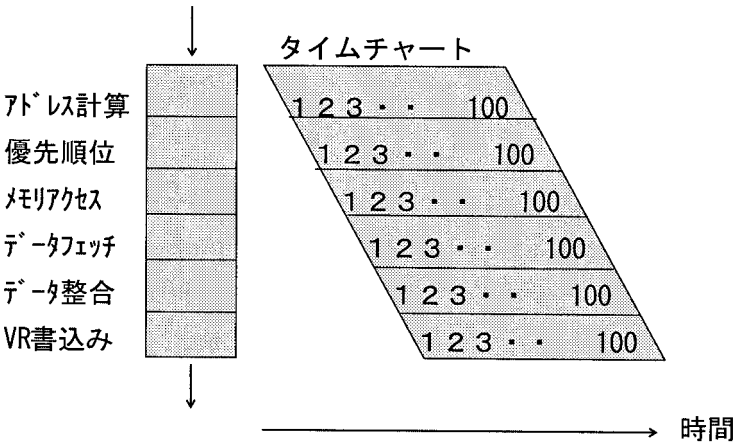
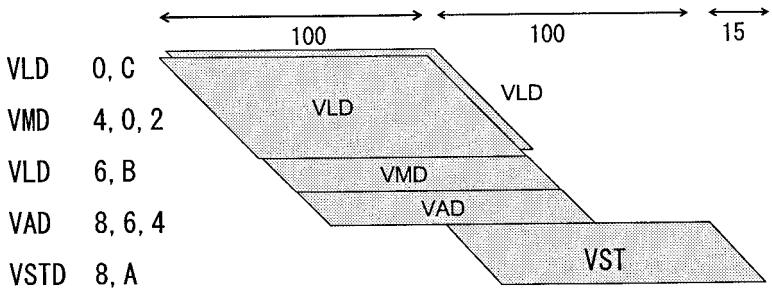


図５　ベクトルロードパイプラインの動作

ベクトル加算命令ではベクトルレジスタに載っているベクトル長分の長さのデータを繰返し演算して，結果をベクトルレジスタに入れる。加算パイプラインでもレジスタの読出し，指数差の計算，指数差分の桁あわせ，仮数の加算，結果の正規化，指数の正規化分の整合，レジスタへの書込みなどのステージを経由して，パイプライン処理が実行される。

さらに，ベクトルプロセッサでは複数のベクトルパイプライン（ロードストア，加算，乗算，除算，マスク演算など）を同時並列に動かして，ベクトル命令間で同時並列実行を行って，性能を引き上げている。

VP-200では，ロードストア，乗算，加算パイプラインを連結して動作させている。この様子を図６に示す。ベクトルレジスタ２にはDが入っているとし，ロード命令によってベクトルデータCがベクトルレジスタ０に入ると乗算命令が起動され，ベクトルレジスタ０をすぐに読み出して入力データとして使う。ロード命令は２本同時に動くことができるので後続するロード命令も起動される。



200+(立上り時間：15) = 215サイクル

図６　DO LOOP例（ベクトル）

ベクトルレジスタ６にデータＢが入ると加算命令が発信可能になる。乗算結果のベクトルレジスタ４とロード命令結果のベクトルレジスタ６にデータが入ると，これらを読み出して加算命令が発信される。図６の菱形は図５の菱形と同じ意味である。ここでは，各命令は１番目のデータが用意されればすぐに命令が発信できる。１番目のベクトルデータから100番目のデータまでは連続して実行されるので図６のようにベクトル命令は100サイクル連続に動作する。したがってロード命令，乗算命令，加算命令はオーバラップして動いている。このことを連結して動作すると言う。加算結果がベクトルレジスタ８に書き込まれるとストア命令は発信可能になるが，ロードストアパイプラインは２本しかないので，ストア命令は，ロードストアパイプラインの空き待ち，すなわち最初のロード命令の動作終了を待って起動される。各ベクトル命令の立ち上がり時間はそれぞれの命令に存在するが，図６では最後のストア命令の分の15サイクルだけが陽に見える。他はパイプライン実行の陰にかくれて見えない。この例をスカラプロセッサで逐次実行すると1500サイクル程度かかるので，ベクトルプロセッサは約７倍の性能を実現できることになる。ただし，上記のサイクル数は説明用であり，実際の数字ではない。図６のプログラムの２番目のロード命令はプログラム上の順序では乗算命令の後で実行することになるが，VP-200では，先行する２命令とは独立に実行できることをハードウェアで検出して，順序入れ替えで追越し実行をしている。これをアウトオブオーダ制御という。この様子も図６に示されている。この制御技術は1970年代初めにスカラプロセッサで試行されたが，それ以降はあまり採用されなかった。最近のスカラプロセッサは採用しているものが多くなってきた。

条件付ベクトル処理は，マスク処理によって実現される。以下の例で解説する。

```
DO  I=1,4
IF(A(I).GT.3)  THEN
E(I)=A(I)+B(I)
ELSE
E(I)=C(I)＊D(I)
ENDIF
END DO
```

IF文はベクトル比較命令によって実行され，大小比較の真偽結果がマスクレジスタR4に置かれている。図７では始めにベクトル加算命令を実行するが，併せてマスク処理機能を同時に行うことを示している。マスク処理機能はベクトル命令の演算結果をマスクレジスタの内容によって真の場合に結果ベクトルレジスタの内容を入れ替え，偽の場合には元の値そのままにする機能である。すなわちベクトルレジスタR2(I)に入っているA(I)とベクトルレジスタR3(I)に入っているB(I)を加算した結果を，マスクレジスタR4(I)の値が1であれば，ベクトルレジスタR1(I)の結果を加算結果に入れ替え，０であればR1(I)の元の値を保持する。その後，ベクトル乗算命令を実行する。

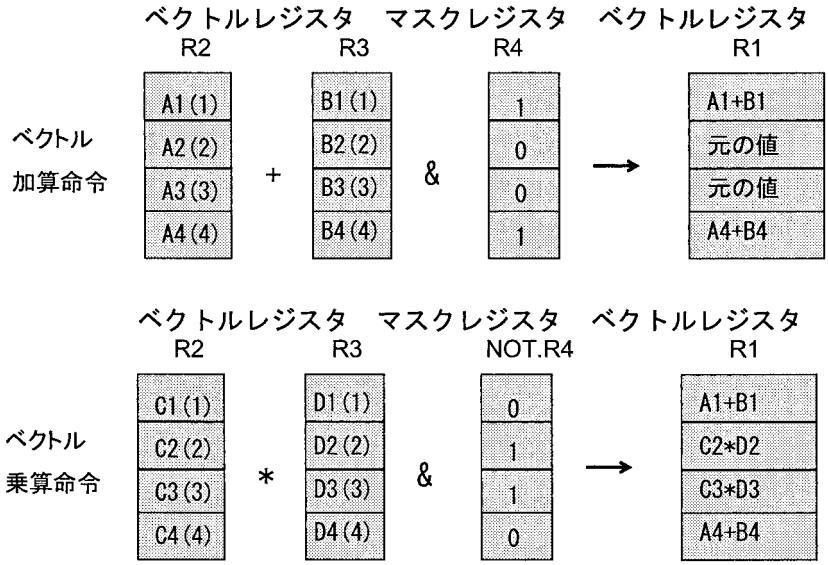


図７　マスク機能

ベクトル乗算命令は，ベクトルレジスタR2(I)に入っているC(I)とベクトルレジスタR3(I)に入っているD(I)とを乗算し，同時にマスク機能を行う。このときはR4(I)に入っている条件の否定（NOT）をマスク条件とする。R1は加算命令と同一レジスタとする。マスク機能によって，R1には先ほどと逆の位置に乗算結果が入り，加算命令で入った値はそのまま保持される。これでベクトルレジスタR1(I)には最終結果E(I)ができ上がる。このようにDO LOOPに条件（IF）文を含むプ

ログラムのベクトル処理はマスク機能で実現できる。このとき注意すべきは，ベクトル演算命令は，すべてのデータを計算してしまうが，条件０の場合計算済みのデータを書き込まないで演算を実行しなかったように振舞うことである。ベクトルプロセッサでは個々のデータごとに演算実行・非実行を制御するより，すべてのIに対するデータを連続的に演算パイプラインに流してしまった方が速いのである。さらに，加算，乗算の２系列の命令を両方とも，投機的に実行してしまい，マスク機能を働かすほうが効率的である。このような投機的命令実行は，命令の並列実行化が進んだ最近のスカラプロセッサにも採用されてきている。以上のように，DOループ内での条件処理（IF文）のベクトル化が可能になったため，大幅な性能向上が得られた。さらなる工夫として， VP-200では，ベクトルパイプラインを物理的に２個並列に置いて，それぞれ偶数・奇数番目のデータを同時実行する方式を採用した。この方式は，後にCRAY X-MPにも適用された。現状のベクトルプロセッサでは，ベクトルパイプラインを複数個，多いものでは16個のベクトルパイプライン演算器を並列同時に動作させている。

ここで説明したようなベクトル処理専用のハードウェアの工夫に加えて，コンパイラによるプログラムの最適化，自動ベクトル化機能の発達により，既存プログラムの画期的な高速が達成可能となったことにより，ベクトルプロセッサは多くの科学技術ユーザに受け入れられた。

３．メモリシステムと高速処理技術

一般的なスカラプロセッサに比べてのベクトルプロセッサの高速性は，先述の複数パイプラインでの命令の多重並列処理とともに，メモリ転送能力の差にあると思われる。

ベクトルプロセッサでは，ベクトル処理用のデータを主記憶装置から直接取ってきて直接書き込む方式を採用している。一方，スカラプロセッサはキャッシュメモリを利用した主記憶アクセスをしているため，データがキャッシュ上にあるとプロセッサの高速性が生きるが，キャッシュ上にデータがない場合には主記憶装置へのアクセスが発生して大きな時間ロスを引き起こしてしまう。ベクトルプロセッサは，このメモリへのアクセスタイム分の遅れ（レイテンシー）を旨く回避できるアーキテクチャである。

ベクトルプロセッサでも，ベクトルロード命令は最初のデータをレジスタに持ってくるのにスカラプロセッサと同等なレイテンシーが発生する。しかし，そのつぎのデータを１クロックサイクル違いで追いかけてデータアクセスを行い，その後つぎ々と１サイクルごとに連続してデータアクセスが生じるので，後続データにはメモリレイテンシー分の遅れは生じない。一連のベクトルロードに対し１回のレイテンシーが発生するだけなので，１データあたりのレイテンシーの影響を減少させることができる。その1データ当りの影響はレイテンシーをベクトルデータ長で割った値となる。図５ではベクトルロード命令ではレイテンシー６サイクルに対し，100個のデータで割った0.06サイクルである。ただし，図6は動作原理を説明するため，レイテンシーを小さくしているが，実マシンのメモリアクセスレイテンシーは30～50サイクルとなる場合もあり，データ１個あたりの影響度は0.3～0.5と大きい。そのうえ，16個のベクトルデータを並列に演算実行している最近のベクトルプロセッサでは，メモリアクセスタイムが480～800個のデータ分のベクトル演

算処理時間に相当する。この影響は非常に大きいので，ベクトル長を大きくしたり，命令の連結実行をしたりしてレイテンシーの影響を吸収しなければならない。したがって，メモリレイテンシーを隠すための，レイテンシー低減・吸収のための技術開発はベクトルプロセッサ設計上の大きな課題である。

一方，一般的にスカラプロセッサでは，キャッシュにデータがないと，データごとにレイテンシーが発生するので，ベクトルプロセッサに比して30～50倍の性能低下になってしまう。したがってスカラプロセッサでも，このメモリレイテンシー回避手段が必要であり，さまざまな工夫がなされている。

さらに，ベクトルプロセッサでは飛び飛びの一定のアドレス間隔によりデータをアクセスするストライドベクトルロードストアや，間接アドレスによるベクトルロードストア（リストベクトル）用に専用処理回路を持っている。個々のベクトルデータを個別にアクセスするための工夫が専用回路として用意されているので，スカラプロセッサに対して性能向上が大きい。

ベクトルプロセッサでは，メモリとベクトルレジスタ間で，ベクトルデータを直接に転送するため，メモリ転送スループットが大きくなければならない。メモリ転送量を単純に増大させることは，バスの太さや本数といったハードウェアの増大につながる。設計者から見れば，ベクトルデータを極力ベクトルレジスタに保持し，主記憶へのアクセスを減少させることができれば，性能向上だけでなく，必要なハードウェアを少なくでき，設計上の利点が大きい。このような発想で設計されたVP-200のベクトルレジスタは，レジスタ番号でプログラム指定可能なベクトルデータ用の大容量キャッシュのような位置付けであり，性能改善に大きな効果がある。

とはいえ，最近のベクトルプロセッサでは，上記のように16個のベクトルパイプラインを同時実行させるため，ベクトルロード・ストアパイプラインのデータ転送能力を極端に大きくする必要がある。演算性能を16GFLOPSと仮定すると64ビット演算では最低16×８バイト＝128GB/Sのスループットが必要になる。メモリのサイクルタイムを50nsと仮定すると８バイト幅のメモリバンクを320個必要とする。メモリバンクの個数は２の冪乗の数が回路構成上，都合が良いので512バンクのメモリインターリーブが必要である。それぞれのベクトルプロセッサごとに，これだけの物量が必要である。共有メモリ型のシステムでは，さらに数倍のインターリーブが必要なため，16個の並列パイプラインはベクトルプロセッサとしては上限であり，８個以内がコストのバランス上，適当と思われる。

VP100/200の方式設計時には，スカラユニット上のキャッシュデータをベクトルユニット側で使うかどうかの問題に直面した。結果としてはベクトルアクセス時には主記憶装置に直接アクセスして，キャッシュを使用しないことにした。この方式の場合は，ベクトルデータをメモリにストアする際，スカラユニットのキャッシュの中身が古いまま残る可能性があり，キャッシュ上のデータと主記憶上のデータに不一致が発生することがある。このとき，当該キャッシュデータを入れ替えるか無効化する必要がある。この動作が頻発すると，ベクトル命令の実行はキャッシュ一致動作のための待ちが生じ，大幅な性能低下が発生する。しかしながら，コンパイラがベクトルデータとスカラデータとを分離して処理するように工夫されたため，この問題は結果的には解

決することができた。その後は他社製も含めてベクトルプロセッサではスカラ処理用データはキャッシュを利用し，ベクトルデータは直接メモリから取ってきてベクトルレジスタに保管する使い分けが一般化した。

Ⅴ．スカラプロセッサの発展（ILP）

１．CISCとRISC

スカラプロセッサアーキテクチャの進展の過程において，できるだけ複雑な演算を一度に実行すれば，実効的に性能向上が得られるとの考えかたで，CISC（Complex Instruction Set Architecture）が一時流行したことがある。しかし，CISCの命令は，簡単な計算機の命令を一度に一個の命令で済まそうとするので，命令処理に多量のハードウェア回路（資源）が必要で，パイプライン化等による並列処理が困難であった。

CISCに対して，RISC（Reduced Instruction Set Architecture）は，複雑な命令を排するかわりに，ほとんどの命令実行時間を１クロックサイクルにした。このため，計算に必要な命令数は増加するが，命令の機能が限られているので先行制御が容易で，命令パイプラインが機能しやすいので，命令実行サイクル数の削減に繋がる。RISCが広まることによって，パイプライン制御方式が発展し，その発展形として，複数命令の同時実行を行うスーパースカラアーキテクチャが始まった。現状では，RISC/CISCの命令制御方式がともに進化した結果，どちらでも高速化が実現できるようになり， CISC/RISC論争は下火になっている。スカラプロセッサの設計におけるメモリアクセスの効率化についていえば，レジスタの数が32から128個程度と増加し，キャッシュと併せて，メモリアクセス転送率の軽減とともに，高価なハードウェアの削減に寄与している。

２．スーパースカラ方式

IBMの801が２命令実行の先鞭を切った。Motrolaの68000もこれに続き，Instruction per Cycle（IPC）が１を超える時代が始まった。これらの計算機方式をスーパースカラアーキテクチャと言う。スーパースカラ方式では，逐次処理も可能な命令の流れを，単純に通常の順序で連続的に命令を複数個ずつ持ってきて実行するものである。図８ではハードウェアは４個の命令を命令キャッシュから命令レジスタに持ってきて，それぞれの命令を演算器に対応する命令発信レジスタの該当する部分に送り込む。命令レジスタの０番目に整数命令が入っていれば，これを命令発信レジスタの整数演算部分に転送する。命令レジスタの１番目に分岐命令が入っていたときには命令発信レジスタの分岐演算部分に転送する。つまりクロスバ回路を経由して，該当演算部分に転送する。このとき演算回路や，レジスタなどぶつかって発信ができない条件が発生すると，そのぶつかり合い条件が解除されるまで，命令レジスタで待っていなければならない。このように，ハードウェアが発信命令の数，種類，待ちなどを考慮して命令実行順序を動的に制御する。

スーパースカラ方式のプロセッサでは，資源のぶつかり合いが発生しないようにすれば，命令の同時実行率があがり，高性能化が達成可能である。命令同時実行率を向上させるためには，コンパイラが命令の並びを旨く配置する命令スケジューリングに対する技術向上が必須である。た

だし，スーパースカラ方式の長所は，逐次処理用のコンパイラで作られた既存のプログラムでも，そのまま実行可能な点である。実行効率は，専用コンパイラでのオブジェクトほどには保証されないが，逐次型のプログラムでもある程度の並列度が達成される場合がある。この場合，ベクトルプロセッサの項で説明した命令のアウトオブオーダ制御が命令の並列実行数を上げるために効果を発揮する。

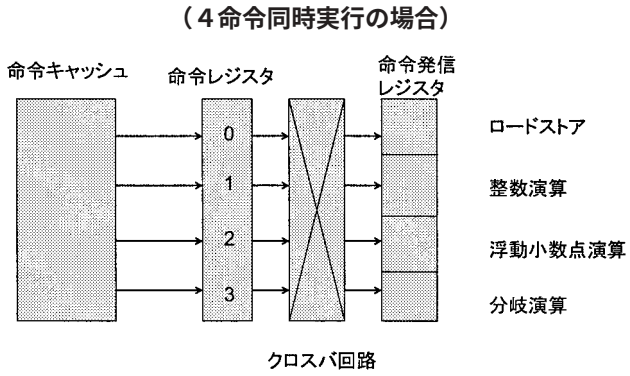


図８　スーパスカラ方式

この10年の間に，1960年代にスカラプロセッサで始まり，ベクトルプロセッサアーキテクチャが育て上げた命令先行実行のための多くの技術は，またスーパースカラアーキテクチャに採用されるようになった。つまり，ある意味ではスカラアーキテクチャ技術はベクトルアーキテクチャ技術をキャッチアップしたと言うことができ，さらに一部はこれを凌駕している。

現状では，スーパースカラアーキテクチャでも，命令の並列実行については，すでに６命令並列を達成し，８命令並列に突入するかに見える。しかしスカラプロセッサアーキテクチャ上の並列化は８命令を超すと効率が悪くなると思われる。それはベクトルプロセッサのベクトルパイプラインも８を超えると効率が低下し，16それ以上にはなりそうにないのと似たような状況で，多重並列処理には限界があると思われる。

３．VLIW

一方，VLIW（Very Long Instruction Word）というアーキテクチャはすでに20年以上も前から，その原型が存在しており，京都大学などでは先進的な計算機の研究が行われていた。VLIWでは，その名のとおり，大きなビット幅を持つ命令語の中に，複数の演算指定部分を持ち，単一クロックサイクル内に複数の演算動作を実行する。

VLIWのコンパイラでは，命令動作の指定及びレジスタの指定をすべて静的に行うことになる。図９に，VLIWで，４演算を同時実行する場合を示す。命令レジスタで指定された命令語はそのまま命令発信レジスタに送られる。したがって命令レジスタの０，１，２，３部分にはそれぞれロードストア，整数演算，不動小数点演算，分岐演算を指定し，かつ，レジスタ番号を解析して，資源のぶつかり合いを避けておかなければならない。ここで資源の競合が発生する場合は，部分

的に無効演算（NOP：No Operation）を指定することになるが，これは並列効率を下げる。

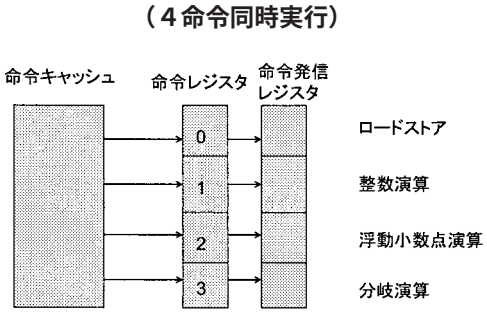


図９　VLIW

VLIWでは命令語の中へ，命令を詰め込まなくてはいけない。この命令のパッキングをコンパイラが行わないといけないので，VLIWは並列処理向けの高度なコンパイラが必須である。また，スーパースカラのように既存の命令の並びを，そのまま実行することができない。

以上のように，プロセッサ内での並列化つまり複数命令同時実行がスカラプロセッサでも可能となったのは，並列処理用コンパイラの技術発展に依存している。

４．スカラプロセッサでの分岐命令処理

スカラプロセッサの性能ネックは，分岐命令の処理である。分岐命令は，全プログラム中に１割から３割程度の割合で存在し，そのかなりの部分が条件つき分岐命令である。ILP方式では，分岐命令処理の高速化の必要性がさらに大きくなる。

単純に条件付分岐命令を実行すると，条件判定が終わるまで，つぎの命令の読出しができないことになる。この問題を克服する方法として，分岐先の真偽両方の命令列を先行実行しておき，最終結果によって，条件に合ったほうの命令の流れを採用し，他方を捨ててしまう方式が考えられた。これを投機的命令実行と言うことがある。

さらに分岐命令の高速化用に，分岐命令履歴表をハードウェア実装することがある。分岐先命令アドレスや分岐の成功不成功の履歴などを記憶しておき，分岐命令に続く命令の流れを予測する。これにより分岐命令処理が高速化された。この分岐命令履歴表は，分岐命令のアドレス，分岐先命令アドレスと分岐成功・不成功などの情報を保持し，分岐命令アドレスをキーにして検索する。この履歴表はキャッシュと同様に連想記憶方式を利用している。ただし，予想が外れた場合には，ペナルティ（性能低下）が大きくなる。

５．マルチスレッド方式

さらにもうひとつのプロセッサ内並列処理方式としてマルチスレッド方式がある。これは，命令の並列実行方式でも，単一プログラム内の命令の並列実行ではなく，ひとつのプログラムを，ひとつのスレッド方式に対応させ，複数のスレッドをハードウェア上で多重実行する方式である。

この方式は1980年代にDenelcoreのHEPという機械に実装された。

この方式では，価格性能向上比が，それほど有利でなかった。つまり，レジスタを始めとして，プログラムごとにプロセッサ資源をスレッド数の分だけ用意する必要があり，レジスタを多数持つアーキテクチャでは，その多重度分，例えば４の場合には４倍の物量を持たなければならない。１スレッド用に128個レジスタを持つ場合は512個のレジスタが必要である。共通に使用できる，例えば演算器などは節約できるが，その相対量は，さほど大きくない。さらに多重スレッド用の多量のハード資源，例えばレジスタから，特定スレッド用の資源を選択する必要があり，多量の資源から必要分を選択するための回路自身が大きくなるだけでなく，アクセスする線長が長くなり，クロックサイクルを遅くしてしまう可能性がある。したがってマルチスレッド方式の有効性は，CPUコアを簡略化して複数CPUを１LSIチップ内に実装するマルチCPUコア方式とどちらが有利かというトレードオフになろう。

マルチスレッド方式では，マルチプログラムの全体（スループット）性能は改良されるが，複数のスレッドにハードウェア資源が割当てられるため，他のスレッドに資源を奪われて，単体プログラム性能はかえって遅くなることがある。また，単体プログラムが，資源を有効利用するためには，プログラムをマルチスレッド環境で動作させるための並列処理コンパイラが必要になり，結局は，マルチプロセッサシステムと大差がなくなる可能性も大きい。したがって，科学技術計算では，マルチスレッド方式は，マルチCPUコア方式に比べてあまり長所は見出せないように思われる。ただし，マルチスレッドと命令レベル並列を混在させるような研究も行われているので，今後の推移を見ていく必要がある。

６．大規模データとキャッシュミス

大規模科学技術計算においては，スカラプロセッサの本質的な短所が，いまだ克服しきれていない。それはキャッシュに入りきらないような大規模データを扱う場合のキャッシュミスによる大幅な性能低下である。キャッシュは単純であるがため，偉大な発明であるが，キャッシュ内データの再使用ができないと無力である。キャッシュミスを検出してから主記憶へのアクセスを起動するようなオンデマンドキャッシュ制御では，常にキャッシュミスを起こした場合に数10倍を超える性能低下を生じてしまう。つまり，ベクトルアーキテクチャのところで述べた主記憶装置との間のアクセスタイムつまりレイテンシーの克服ができないわけである。これを脱却する適切な解は何十年もの間見出されなかった。

現在のスカラプロセッサのクロック周波数は２GHZを超え，つまりクロック周期は500psを下回った。一方メモリデバイスの読出し書込みサイクルはいまだに数10nsであり，100倍前後の差がある。したがって，メモリ階層としてのキャッシュと主記憶間の速度の乖離ははなはだしく，10年前に比べてかえって大きくなっている。

通常のプロセッサ（例えばPCなどのコモディティCPU）では，メモリアクセスの高速化は，キャッシュにより，ユーザ空間の大容量化はDRAMチップにより実現され，プログラムデータ空間が小さい一般のユーザ要望を満足したと言って良い。しかし，スーパーコンピュータを使用する

ようなHPC（High Performance Computing）領域のユーザ要求を満足していない。これを改善するためには，メモリシステムの高速化をメモリデバイス開発者が実現するか，システム構成上のアーキテクチャの工夫を計算機アーキテクトが解決するかしなければならない。残念ながら，メモリデバイスは，PCやゲーム機など大量消費市場からの要請として，コスト削減と大容量化の方がはるかに要求度は高く，高速化の要求は二のつぎである。したがって，現状では計算機アーキテクトが構成上の工夫によって何とかこれを解決しなければならない。

７．キャッシュミスの克服

この大問題を克服できる解の可能性は，ハードウェアアーキテクチャとしてオンデマンドキャッシュ制御に代わる技術を開発することであり，また，コンパイラ技術によって，処理すべきデータが常にキャッシュ上にあるようにする工夫である。ともに単独では，すべてに対応できないように思われ，双方が協調しなければならない。

ハードウェアアーキテクチャによって解決する方式として，レイテンシーを隠すメモリアクセス方式（レイテンシー克服方式：Latency Tolerant Memory Access）がある。ベクトルロードに同じように，最初のメモリレイテンシーは省略できなくても，以降のメモリアクセスでメモリレイテンシーを隠してしまえばよい。これを実現するためには，プリフェッチ命令の利用が考えられる。プリフェッチ命令は，キャッシュにデータを持ってくることを目的として，ロード命令に近い機能を実行するが，データをキャッシュにおくだけでレジスタに書き込まない。プリフェッチ機構を利用して，メモリアクセスを発信し，実際に使用するより前にキャッシュにデータを読出しておく。プリフェッチ機構は，通常のCPU演算とは独立に動作が可能であり，CPU命令実行には影響を与えないように設計することができる。ベクトルロード命令がメモリアクセスをDOループの先頭で繰返しパイプライン的に発信するように，プリフェッチ命令で配列データをあらかじめキャッシュに持ってきてしまうのである。プリフェッチされたデータが，他の命令の実行のためにキャッシュから追い出されなければ，キャッシュミスは大幅に削減でき，データキャッシュはベクトルプロセッサのベクトルレジスタと同等な位置づけとすることができる。

現在のスカラプロセッサのキャッシュは１次キャッシュがすでにベクトルレジスタと同レベルの容量を持つようになっており，２次キャッシュを含めれば，はるかに大きな容量になっている。したがって，適正な大きさの配列を規定すれば，キャッシュから追い出される確率を小さくできる。ベクトルレジスタ程度の容量を想定し，ベクトルコンパイラがベクトルロードで実現しているようなメモリアクセスするように，ベクトルコンパイル技術を利用して，複数のプリフェッチ命令を発信することができれば，ベクトルプロセッサで成し遂げたメモリレイテンシーの克服が実現できる。ベクトル命令では，メモリアクセスは，１命令の発信だけで済むが，今考えているプリフェッチ方式では，全配列をキャッシュのブロックサイズで割った回数の命令発信が必要である。ハードウェアが自動プリフェッチ機能を持っていれば，その分プリフェッチ命令数の発信を少なくすることもできる。本案では，スカラプロセッサ自身がベクトルプロセッサと同等に多数のメモリアクセスをメモリアクセスパイプラインで処理できる強力なメモリシステムを持つこ

とが必要であり，さらにベクトルコンパイラ技術をスカラプロセッサコンパイラにも実装する必要がある。ストライドアクセスベクトルやリストベクトルと同等なメモリアクセスパタンでは大容量２次キャッシュへのデータ保持ができるようなコンパイルが期待される。ストライドアクセスでは多重ループの外側のDOループで同一ブロック内データが繰返し使われる可能性が高いので保持したデータが有効に利用できる。

ソフトウェアによる案は，大規模配列のメモリ領域を細分化し，キャッシュ容量以下の局所領域に限定できるようにして，キャッシュミスを低減する方式である。この方式は，並列処理向けのコンパイル技術の中で，一つのプロセッサあたりにデータ領域を局所化する方式と共通技術として実現できる可能性がある。そのためには，大規模データの局所化と，そのデータを用いてできる計算の範囲及び実行時間を考慮した命令スケジューリング技術を確立しなければならない。上記のストライド，リストアクセスも，この手法を応用できると思われる。以上のようなレイテンシー克服（Latency Tolerant Memory access）方式は富士通Prime Power HPC 2500システムのハードウェアとコンパイラに実装されつつある。今後この技術は評価され，より高い実効性能を目指し，改善され成熟化していくことが期待される。

今後のスカラ並列システムには，ベクトルコンパイラ技術も並列コンパイラ技術も活用して，メモリレイテンシーを隠してしまうための処理技術と併せて，処理の細分化とデータの局所化を目指した並列化技術を利用する総合的なアプローチが必要であろう。

．並列プロセッサシステムの発展

１．ハードウェア

高速化と並列化のところで述べたように，35年前から多くのシステムが開発されてきた。1980年後半からはいわゆるMPPシステムが雨後の竹の子のように出現し，数年で消えていった。最近ではクラスタシステムが出現し，SMP（Shared Memory System：共有メモリシステム）をクラスタとして，それを並列化していく多重レベルマルチプロセッサシステムが商品化されている。この方式の構成には，SMPクラスタに多数のプロセッサを配置する場合と４プロセッサ程度の少数台に限定する場合とがある。SMPにはUMA（Uniform Memory Access）とNUMA（Non Uniform Memory Access）がある。UMAではメモリとの距離を等長にする必要があり，多くのプロセッサを組み込むにはコストが高くなる。NUMA方式はコストが低いが，プロセッサとメモリが等距離でないため，処理のアンバランスが発生しやすく，動的なジョブのスケジューリングがさらに複雑になる。

SMPにせよ，DMP（Distributed Memory System：分散メモリシステム）にせよ，ローカルメモリ割り当て論理は複雑である。前に述べたキャッシュミスを避けることを考慮したプログラムの計算処理とデータの割り当てについては，コンパイラが多くのトレードオフを解決しなければならない。

クラスタ間のネットワークとそのトポロジーについては多くの研究が行われており，理論的解明も行われている。クロスバネットワークはすべてのクラスタ間の結合が可能であり，結合性を

絞った各種の多段ネットワークもある。具体的アプリケーションでは，それぞれのトポロジーで適合しやすいものもあるし，不適当なものもある。

特に同期処理に必要な時間と処理の粒度（処理のまとまりと経過時間）とには，大きな関係がある。粒度が大きいとシステムコストは低いがアプリケーションの範囲が狭まり，粒度を小さくするとアプリケーションは広がるがシステムコストは高くなる。

２．ソフトウェア

並列処理用のソフトウェアの発展は目覚しい。OSはマルチスレッドをサポートしており，SMPシステムでの並列処理効率が大幅に改善されてきた。並列処理用の言語（OpenMP，HPF（High Performance Fortran））及びライブラリ（MPI（Message Passing Interface））の仕様は標準化された。OpenMPは主にSMPシステム用に，HPFはDMPに使われている。MPIはSMP，DMP両方に使われている。

DMPシステム用の言語としては，データ並列型のHPFとメッセージパシングインタフェースのMPIがある。データ並列型の言語としては，NWT-FORTRAN，VPP-FORTRANなど機械固有の言語も多いが，HPFと同様な目的をもった仲間関係にある。

HPFはユーザ指定Directive（指示行）を持つ。Directiveの重要な機能は，個々のプロセッサメモリへのデータの割付けである。Directiveを挿入することにより，同期処理など並列処理サポートのかなりの部分をコンパイラと実行時ライブラリがやってくれる。HPFは３者の中では，もっともプログラミングのユーザ負担が少なく，システムソフトウェアに任される機能が多いが，HPFをサポートする商用システムの数が少ないため，あまり広まっていない。すでに世の中の言語としてはCが広まりつつあり，FORTRANでは多くのプログラマを確保できないのかもしれない。あるいは個別機械の言語が広まっているのでHPFを使わないと言えるかも知れない。

OpenMPは，やはりDirectiveで並列処理機能をサポートするが，同期処理もユーザの負担である。

MPIは並列処理用のシステムライブラリであり，すべての機能はユーザ管理であるので，同期処理はもとより，すべてユーザが管理しなければならない。

OpenMPとMPIは，HPFよりプリミティブなソフトウェアであるのに，ポピュラーであり，より高級な言語のHPFの方が少数派であるのは面白い。

さらにSMPシステムでは，FORTRANやCコンパイラによる自動並列化が実現できている。少数台のマルチプロセッサシステムでは，並列効率もそれなりに出ている。

．性能評価とアプリケーション

性能の評価には標準的なベンチマークプログラムがあり，ある程度のシステムの評価に使えるが，ユーザ固有の実アプリケーションプログラムに対する性能でないと本当の性能評価にはならない。あるプログラムの性能Pはその部分プログラム i に対する性能Piの集積によって評価すべきである。性能は演算する総浮動小数点数Miを実行時間Tiで割った数字ＭＦＬＯＰＳi（Million

Floating Point Operations Per Second）で示すことが多い。一方，機械のピーク性能Sが定義されているが，これはあくまですべてのハードウェアが設計のとおりに動いたときの機械の最大性能である。したがってつぎの式が成り立つ。

$$MFLOPSi=Mi/Ti=S\times \alpha i \qquad (1)$$

（αi は単一プロセッサの部分プログラムiの実効演算性能効率）

さらに並列プロセッサでの実効性能Piはつぎのようになる。

$$Pi=Ni\times S\times \alpha i\times \beta i \qquad (2)$$

（Niは部分プログラムiを実行するプロセッサ台数，βiは実効並列処理効率）

全体のアプリケーション性能Pはつぎのようになる。

$$P=\Sigma Pi=\Sigma Ni\times S\times \alpha i\times \beta i \qquad (3)$$

この式では個々の部分プログラムを平均化した性能をあらわすことになる。ただし（3）式は同一の重み付けを仮定したが，さらに部分プログラムに重み付けをすることで，部分プログラムの出現確率を加味させることもできる。このような分析によって典型的な部分プログラムの走行性能を推定し，システム開発に反映させる。

ベンチマークプログラムを決めたあとでは，実際はプログラム全体がどのくらいの時間で走るかが絶対的な性能である。したがってベンチマーク性能評価は単純に以下のようになる。

$$P=1/\Sigma Ti \qquad (4)$$

一般的にこの性能Pを価格で割った数値が価格性能比と呼ばれ，システムを評価する良さの指標となる。

コンピュータアーキテクトは，最終的に，この数値を最大化するために上記の個別の性能評価を徹底して追及し，個別プログラムの重み付けにも適切なノウハウを加味した最適化を図らなければならない。

すでにHPC（High Performance Computing）が並列処理時代に入ってしまったと考えれば，具体的なアプリケーションプログラムでは，物理現象のシミュレーション，流体計算，FFT，有限要素法など，従来からの一般的な科学技術計算ではプロセッサ間の通信がクリティカルになるものが多い。

一方で，データベース検索，DNAの相同性サーチ，解析可能暗号解析，パターンマッチングなどでは，プロセッサ間の通信がクリティカルではない。このようなアプリケーションでは，PCク

ラスタやパーソナルコンピュータを何千，何万台を参加させ処理している場合がある。最近ではこれらの大規模なものが増えてきている。（２），（３）の式で言えば，βi=１に近いアプリケーションなので，プロセッサ間通信として工夫することはなく，計算機アーキテクチャとしてかなり単純化できる。すでにPCクラスタもスーパーコンピュータのひとつとしての位置づけを確保している。

．今後の見通し

１．テクノロジー

いわゆるムーアの法則（１年半で２倍の集積度）は，10年来経験則として成り立ってきた。この法則はいまだに通用しており，今後５年くらいは通用するだろうと言われている。CPUチップの集積度は1000万トランジスタを超え，クロックサイクルは３Ghzになろうとしている。昨今のナノテクノロジの発展は目覚しく，CMOS及びその周辺技術もナノテクノロジの発展に刺激されて，さらに縮小化が促進される気配が濃厚である。

スカラプロセッサの項で述べたように，メモリデバイスの大容量化とともに高速化が実現できないと，プロセッサ全体の性能はメモリ性能で抑えられてしまう。計算機ハードウェア設計者にとっては，この問題の解決が図られることを切に願うものである。ナノテクノロジによって，原子数個単位でのメモリデバイスが実現できれば，速度，容量の両面から革新的な高性能デバイスが実現でき，メモリ性能（アクセスタイム，転送能力）が性能のすべてを決定するスーパーコンピュータとしては，ベクトル型であれ，スカラ型であれ，高速大容量メモリを歓迎するであろう。ただし高性能であってもコストが安くなければ意味がない。PCクラスタへの価格競争にも勝ち残らなければならないからである。

さらに大規模集積化が進めば，システム全体レベルのLSIも可能になり，計算機アーキテクチャへも大きな影響を及ぼすと思われる。

２．計算機アーキテクチャ

ベクトルプロセッサが生き残るためにはコスト革命が必須であろう。製造メーカとしては技術推進力になるといっても，利益が確保できなければ事業としての継続はありえない。とくに高メモリスループットに費やされる回路素子のコスト限界が厳しく，上記の高性能メモリ素子に期待するところが大きい。

スカラプロセッサでもさらにコスト削減が必須であるが，ベクトルプロセッサと同様にメモリスループットの大幅な向上とメモリパイプライン化によるメモリ回路の複雑化による高価格化が吸収されるような革新的技術が必要である。

スーパーコンピュータアーキテクチャにおいては先に述べた，レイテンシー克服方式の成熟が期待される。性能とともにコストは計算機アーキテクチャの大きなドライビングフォースであるため，両者を併せて解決しなければならない。さらにコンパイラ技術との協業が必須である。最終的には総合システムとして価格性能比の良いシステムが生き残ることになる。

一方，外部記憶を見てみると，すでにスーパーコンピュータの主記憶容量が10TB（テラバイト）を超し，２次記憶容量はPB（ペタバイト）を超える時代になっている。ディスク容量の集積密度は上記ムーアの法則以上に急激に改良されている。しかし，大量データ転送でのデータ転送率の不足が大きな問題である。例えばPBクラスのデータのバックアップが１日以上かかるなど，問題は深刻である。この解決案としては，超微細加工技術を２次記憶装置にも適用して，容量増加とともに転送率の改善をしていかなければならない。

プロセッサ性能と主記憶アクセスタイムと同様に，レイテンシーギャップが，主記憶と２次記憶との間に存在する。ディスク装置でのアクセスレイテンシーは微細加工によって少しは改善されるが，そのギャップは増加している。ディスクキャッシュは主記憶と同じように小容量のデータに対してレイテンシーの吸収に大いに役立っている。しかし，大規模データへのレイテンシーの解決は，２次記憶まで含めた記憶スペースの細分化，局所化によって，アプリケーションレベルからの解決が必要であろう。これは結合ネットワークの構成にも深く関係がある。

３．結合ネットワークとシステム実装

プロセッサ及びクラスタ間の結合ネットワークの改善も期待したい。今後は地球シミュレータのように銅線ケーブルを3000km使うような高コストシステムは存在できないと思われる。光ケーブル化することにより，線長制限，重量，転送率の問題が解決できそうであり，結合ネットワークとして最適な素子になると思われる。誤り率の改良は通信用光デバイスの量産化に伴なって成し遂げられると思われる。光技術が発展すると，３次元方向のデータ転送の可能性もあり，実装効率を革新できる可能性もある。縮小実装技術についてはナノテクノロジの革新の進展に伴って発展することが期待でき，論理素子，メモリ素子，入出力機構などシステム全体の縮小化が実現できるであろう。そうすれば線長の短縮によって高速化が達成される。３次元実装の縮小化はシステムアーキテクチャにも影響し，変貌していくことになろう。そのときでもやはり，プロセッサクラスタ間のネットワークレイテンシー問題は存続するように思われる。つまり，絶対的なアクセス時間は減少するであろうが，プロセッサとネットワーク双方が高速化するので，相対的なギャップが課題として残ると思われる。

かつてのように，スーパーコンピュータが計算機技術の基盤を作ることはできないかもしれないが，少なくとも微細化実装の先頭に立つべきであろう。ナノテクノロジーの進展がスーパーコンピュータへ，最終的にはユビキタスコンピューティングへの応用ができる共通技術への貢献を期待したい。

４．グリッド（GRID）

グリッドとは「地域も組織も別で非対称な分散環境で高性能な計算やデータ処理を行うことができるインフラストラクチャ」と定義され，「いつでもどこでも計算パワーが取り出せる環境」を言う。後のグリッドへ発展して行った初期のプロジェクトは1990年代の中ごろから始まった。米国でのたくさんの計算機で計算を共有するMeta Computingプロジェクトや，計算機の使用環境を

どこからでも同じよう（Transparent）に見せることを目的として始まった欧州でのUnicoreプロジェクトなどを契機として，グリッド（コンピューティング）・プロジェクトが世界的に大きな流れとなっている。GGF（Global Grid Forum）が組織され，多くの科学者や技術者がこれに参加している。あきらかにここ数年間はブームが続くと思われるが，中にはInternetと同じくらいの社会インパクトとなると予測する人もいる。グリッドはWeb技術と融合して，より社会に貢献できる主流の技術になることを期待したい。

インターネット・Web上でのデータ転送率は急激に向上し，GB/秒クラスにすでに達しているため，現状に比べれば，さらに快適なネットワーク環境が実現できる。しかし，信号の伝送速度は光速度以上にはならない。グリッドでは地球規模のデータの分散配置，計算リソースの分散配置を目標としている。したがって物理的な距離は縮まらないわけで，地球規模の距離でのデータ転送レイテンシーはシステム内のレイテンシーとは比べものにならないオーダとなる。レイテンシー短縮ができないとすれば，必要データベースを自システムへ取り込んでしまえばよいが，データ転送量と自システムの記憶容量の限度がある。したがって，グリッドレベルでもデータ転送率とともにレイテンシー克服を図っていく必要があろう。グリッド要素内でデータベースとアプリケーションをいかにしてローカライズできるかが課題である。

５．新方式のコンピュータ

未来のコンピュータ技術の一つとして，並列処理を前提にしたDNAコンピュータや量子コンピュータが話題に上っている。並列処理に向いているそれぞれの原理によって計算アルゴリズムが成立する。しかし現在の計算機と同じように使おうと思っても，信頼度，入出力機構，コスト評価など課題は数多いであろう。しかし既存領域以外に応用分野を見出すことが目的であり，そこに価値があるのだと思われる。応用分野を特定すれば，現在の計算機に比べて大幅な性能向上を期待できる可能性がある。逆に，それが既存分野に大きな影響を与えるかも知れず，既存計算機との協調もありうるかもしれない。したがって今後の技術発展を注意深く見守って行く必要がある。

．最後に

名古屋大学情報連携基盤センターからスーパーコンピュータの歴史と今後についてと言う題名で記事を書くようにとのご依頼で気軽に引受けましたが，書くにしたがい，あれもこれもと題材が増え，結構な分量の解説記事になってしまいました。なるべく平易に書いたつもりですが，解説不足のところもあるかと思われます。読者が補っていただけると幸いです。自分の記憶に頼って確認を取らずに記述したところもありますので，誤りがあればご指摘願います。

最後に，名古屋大学情報連携基盤センター長谷川明生助教授，富士通の神谷幸男，向笠滋両氏からいくつものご指摘やコメントをいただきました。ここに感謝いたします。

（うちだ　けいいちろう：神奈川大学理学部情報科学科）