第 9 章
マルチ・ベクトル
— ベクトル記号処理のためのデータ構造

要旨

第 6 章でのべた逐次論理型言語プログラムのベクトル処理法と第 7 章でのべた並列論理型言語プログラムのベクトル処理方法の両方において，共通のデータ構造「マルチ・ベクトル」がつかわれることがわかった．この節ではマルチ・ベクトルということばによりひろい定義をあたえ，論理型言語以外の応用までふくめてマルチ・ベクトルの機能，マルチ・ベクトルに対する操作などについて考察する．すなわち，マルチ・ベクトルはベクトル記号処理の広範な応用において，ベクトル再生成オーバヘッド回避にやくだち，バックトラックや並列処理の処理単位としてはたらき，ベクトル・パイプラインの有効活用にやくだつとかんがえられる．したがって，マルチ・ベクトルはベクトル記号処理において非常に重要なデータ構造であると結論できるだろう．

9.1 はじめに

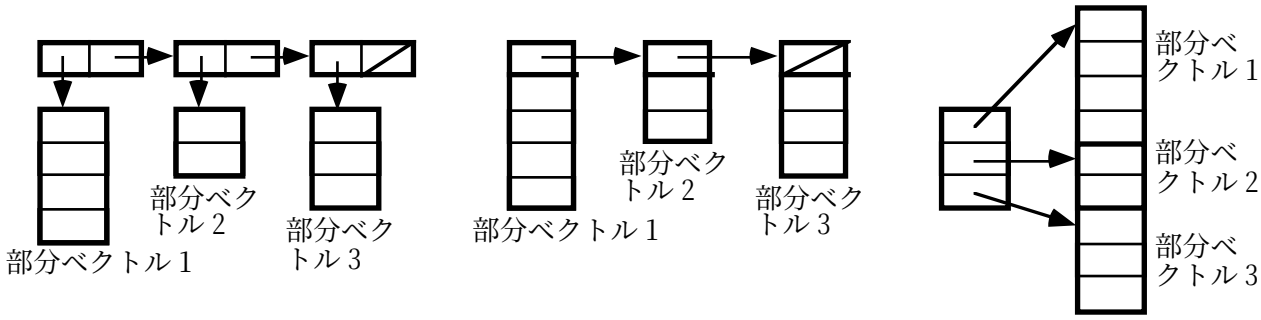
われわれは，Cray-1，HITAC S-810，HITAC M-680H IAP など，数値計算専用機として発展してきたパイプライン型ベクトル計算機をリスト，木，グラフなどの可変データ構造をあつかう記号処理 (非数値処理) に適用し，その高速実行をはかってきた．また，データベース処理用のベクトル計算機である M-680H IDP をデータベース以外の記号処理に応用する実験をおこなってきた．記号処理プログラムにおけるリスト，木，グラフなどの可変データ構造は，そのままではベクトル計算機によって実行できないばあいや高速には実行できないばあいがしばしば存在する．このようなプログラムを高速にベクトル処理できるようにするためには，ベクトル計算機に適合したデータ構造をみいだし，もとのデータ構造をその構造に変換することが重要である．われわれが開発した上記の論理型言語の OR 並列処理方式と AND 並列処理方式もこのようなデータ構造変換にもとづいている．

ところが，これらの実行方式はまったくことなるベクトル化方式にもとづいているにもかかわらず，その実行においてほとんどおなじデータ構造がつかわれ，そのデータ構造がよく似た操作をうけていることがわかった．また，類似のデータ構造が M-680H IDP におけるマージ・ソートにもつかわれていることがわかる．これらのデータ構造をわれわれはマルチ・ベクトルとよんでいる．マルチ・ベクトルの応用法をしめし，その機能を分析するのが，この章のおもな目的である．

9.2 節ではマルチ・ベクトルの定義をしめす．9.3 節ではマルチ・ベクトルがベクトル記号処理においてどのように使用されるかを例をつかってしめす．9.4 節では，9.3 節でしめした応用例においてマルチ・ベクトルがはたしている役割を分析する．9.5 節ではマルチ・ベクトルに対する操作としてどのようなものがあるかをしめす．

9.2 マルチ・ベクトル

複数個の可変長ベクトルまたは固定長ベクトルをなんらかの方法で連結したデータ構造を**マルチ・ベクトル**とよぶ．ただし，各ベクトルは同質の値をふくむことを仮定する[注1]．マルチ・ベクトルを構成する各ベクトルを**部分ベクトル**とよぶ．**図** 9.1 (a) にしめすようなベクトルを要素とするリストをマルチ・ベクトルの基本形とかんがえる．Prolog で容易に記述できるなどの理由によって，第 6 〜 7 章では基本形のマルチ・ベクトルを使用している．しかし，より低水準の言語で記述するばあいには，部分ベクトルの先頭要素でつぎの部分ベクトルをさす図 9.1 (b) のようなかたちのほうが，時間，記憶のいずれにおいてもより効率がよいであろう．また，図 9.1 (c) にしめすように各部分ベクトルへのポインタを要素とするベクトルもマルチ・ベクトルの一種だとかんがえることができる．さらに，図 9.1 にしめした以外にもマルチ・ベクトルのさまざまな変形をかんがえることができる．なお，(a)，(b) のように可変長の部分ベクトルを連結したばあいには，処理の際に各部分ベクトル要素数を知る必要があるので，その値を部分ベクトル先頭などに格納する必要がある．



(a) **基本形**　　(b) **最適化形**　　(c) **変形**

図 9.1　**マルチ・ベクトルとその変形**

[注1] ここで「同質」ということばにはあいまいさがあるため，マルチ・ベクトルの定義にもあいまいさがふくまれることがとりあえずはさけられない．しかし，どのような範囲のデータ構造をマルチ・ベクトルと定義するのが適切であるかがまだ明確でないので，あえて厳密な定義はあたえないことにする．

9.3 マルチ・ベクトルの応用

現在わかっている，ベクトル記号処理におけるマルチ・ベクトルないしそれに類似のデータ構造の使用法には 3 とおりある．第 1 は エイト・クウィーン問題のような解探索において部分解を格納するという用途である．第 2 は素数生成問題のようなストリーム処理においてストリームの要素を格納するという用途である．第 3 はマージ・ソートである．これらについて順に説明する[注2]．

9.3.1 解探索における使用

第 6 章では，Prolog のような論理型言語で記述されたバックトラックをつかった解探索プログラムを，ベクトル化するすなわちベクトル処理できるようにかきかえるばあいにマルチ・ベクトルを使用し，エイト・クウィーン問題のベクトル化されたプログラムの実行性能をしめした．第 6 章でしめした方法においては，もとのプログラムにおける複数のことなる入力データまたは出力データ (すなわち解) を要素とするベクトルを使用し，もとのプログラムではバックトラックしながらくりかえし処理してもとめていた複数の解を一度にもとめる．そして，**図 9.2** に例示するように，ひとつの入力からえられることなる解に関する計算すなわち OR 関係にある複数の計算を複数の入力データをふくむ 1 個のベクトルに対して独立に計算し，それぞれの計算から複数の出力ベクトルを生成する．図 9.2 は `carcdr` という Prolog の述語のベクトル処理の様子をしめしている．このばあいには，図にしめした `carcdr` の 2 つの節の計算が `[a|b]`，`[c|d]` という入力データに関して独立に実行され，独立に結果をえている．これらの節が出力するベクトルをまとめてあつかうためにつなぎあわされ，その結果として自然にマルチ・ベクトルが生成される．

ただし，正確にいえば図 9.2 においては入力したベクトルもただひとつの部分ベクトルからなるマルチ・ベクトルの形式をしている．したがって，入力とくらべて出力においてはマルチ・ベクトルを構成する部分ベクトルの数がふえているだけである．このように部分ベクトルが 1 個のときにもマルチ・ベクトルを使用することにより，入力するベクトルが 1 個のときも複数個のときも同一の入出力インタフェースですむようになり，したがってひとつのプログラムで両方のばあいをあつかうことができるようになる．

6.4 節でしめした完全 OR ベクトル化の方法においては，マルチ・ベクトルの生成がおわるとただちに**図 9.3** のようにしてただひとつのベクトルに併合される (**部分ベクトルの併合**)．しかし，6.5 節でしめした並列バックトラック化においてはマルチ・ベクトルはすぐには併合されず，バックトラックの単位として使用される．部分ベクトルの併合に

[注2] なお，マルチ・ベクトルによく似たデータ構造として6.10 節でしめしたフレームがある．しかし，フレームは部分ベクトルが同質の値をふくむとはいえないので，マルチ・ベクトルとはみなさないことにする．

関しては 9.6 節でくわしく説明する．

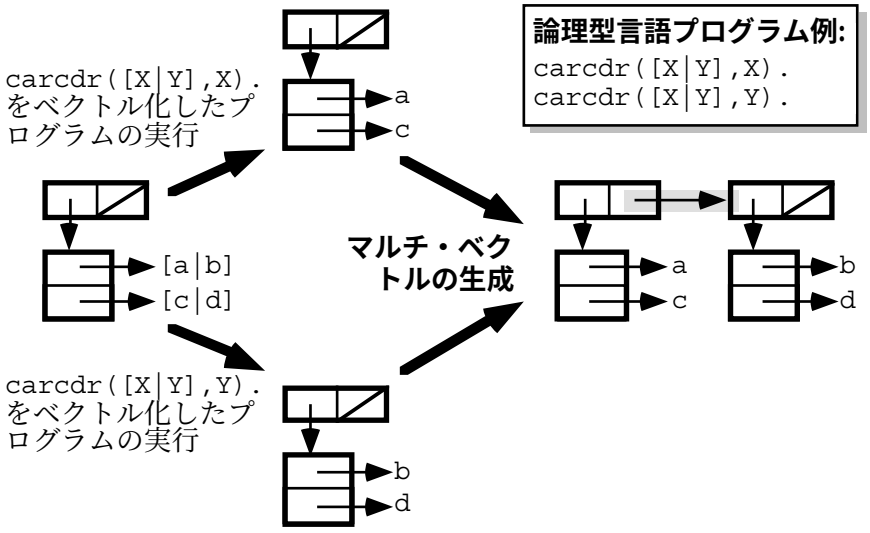


図 9.2　解探索におけるマルチ・ベクトルの生成

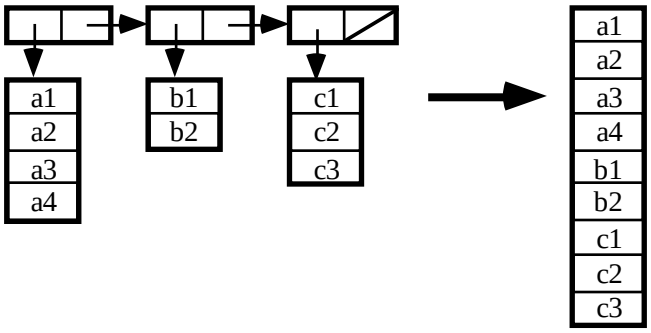


図 9.3　部分ベクトル併合の基本

9.3.2 ストリーム処理における使用

第 7 章では，逐次論理型言語 Prolog および GHC による素数生成のプログラムを手動で変換してベクトル処理可能なプログラムを生成し，その性能測定結果をしめした．このベクトル化法は任意の論理型言語によるプログラムに適用できるわけではない．しかし，論理型言語で記述されたプログラムにかぎらず，ストリーム処理，いいかえればフィルタリング処理においてはばひろくつかえるものだとかんがえられる．

素数生成を例として，ストリーム処理におけるマルチ・ベクトルのつかいかたを**図** 9.4 に例示する[注3]．ここでは，整数列をふくむマルチ・ベクトルに対する素数の倍数のフィルタリング処理のくりかえしの結果として素数が生成される．フィルタすべき整数列のながさ N が非常におおきいばあいには，これらをひとつのベクトルにいれて一度に処理するのは得策でない．なぜなら，この処理には，入出力ベクトルのほかにもすくなくとも $O(N)$ の膨大な作業記憶が必要だからである．また，整数列を複数のばらばらのベク

[注3] ただし，この図は第 7 章における処理をずっと簡略化したかたちでしめしている．

トルにわけていれておくと，管理が容易でない．したがって，整数列をマルチ・ベクトルのかたちにするのがベクトル処理のために適切だということができる．

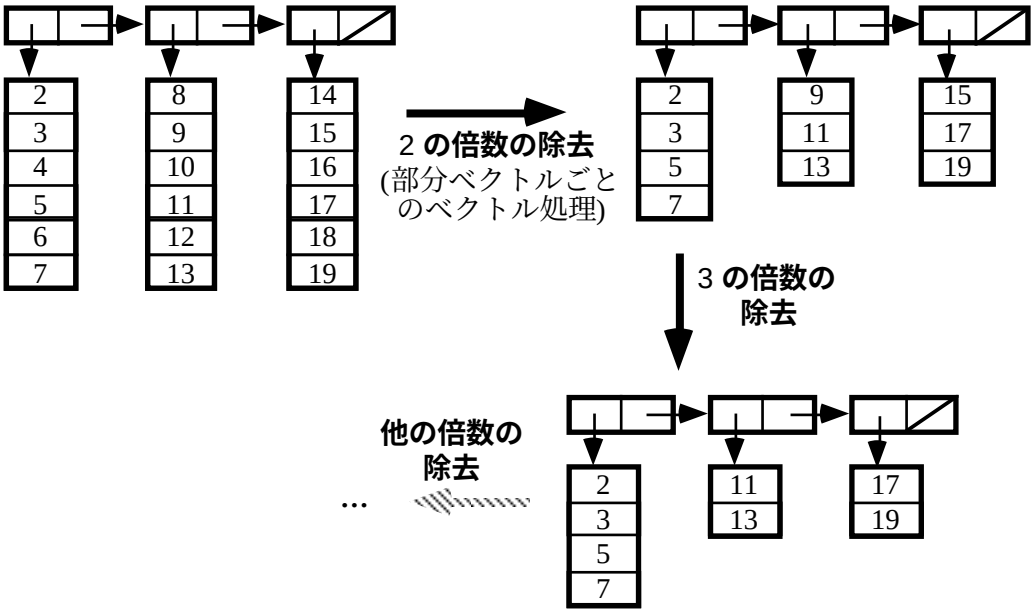


図 9.4　素数生成におけるマルチ・ベクトルの使用

素数生成のばあいは，フィルタリングがすすむと，図 9.4 にしめしたように部分ベクトル要素数が減少していく．要素数がすくなくなりすぎるとベクトル処理性能が低下する．したがって，第 7 章でしめしたように，適当なタイミングで部分ベクトルの併合をおこなうことにより，性能を改善することができる．

9.3.3 ソートにおける使用

M-680H IDP においても，マージ・ソートをおこなう際にマルチ・ベクトルにちかいデータ構造が使用される (**図** 9.5)．ただしこのばあいには，図 9.5 でハッチングをほどこした部分ベクトルの先頭をさすベクトルは実際には生成されない．なぜなら，各部分ベクトルの先頭アドレスは最初の部分ベクトルの先頭アドレスから計算することができるからである．各部分ベクトルの先頭アドレスが計算できるのは，全要素数が 2 のべき乗のばあいはこの疑似マルチ・ベクトルのすべての部分ベクトルの要素数が一定であり，2 のべき乗でないでないばあいでも最後の部分ベクトルをのぞけば各部分ベクトルの要素数は一定になるからである．しかし，利点があるかどうかはべつとして，部分ベクトルの要素数が一定にならないソート方法は容易につくることができる．このようなソート方法をベクトル処理しようとすればハッチングした部分も必要になり，図 9.1 (c) の形式のマルチ・ベクトルが必要になる．マージ・ソートにかぎらず，計算時間が $O(N \log N)$ であるソート方法をベクトル処理しようとすれば，マルチ・ベクトルまたはそれにちか

いデータ構造が必要になるとかんがえられる.

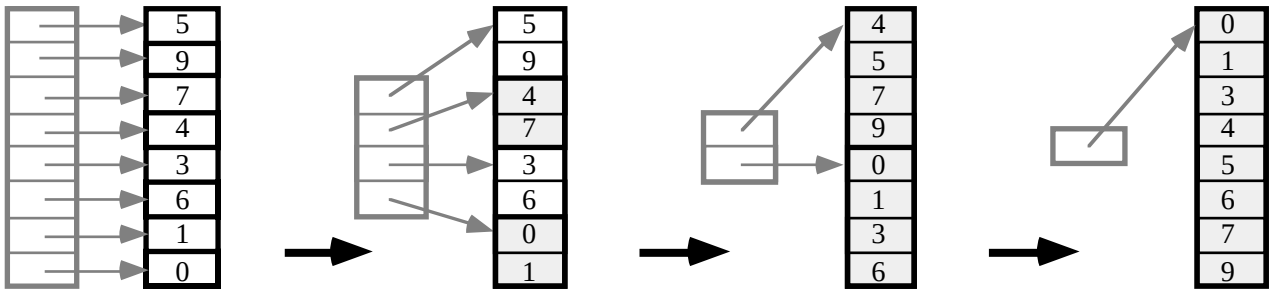


図 9.5　M-680H IDP におけるマージ・ソートの方法

9.4 マルチ・ベクトルの機能

マルチ・ベクトルは，第 1 にベクトルの要素数が変化する際にベクトルを再生成するオーバヘッドをなくすはたらきをする．また第 2 に，バックトラックや並列処理における処理単位という役割をはたす．さらに第 3 に，ベクトル計算機のパイプラインをより有効に利用できるようにする．マルチ・ベクトルがもつこれらの機能について説明する．

9.4.1 ベクトル再生成オーバヘッドの回避

解探索においてマルチ・ベクトルを使用するおもな理由は，ベクトル再生成による実行時間や記憶消費量のオーバヘッドをさけることにある．数値計算であつかうベクトルは，処理がすすんでも通常は要素数が一定である．これに対して，記号処理におけるデータ構造をベクトルに変換したばあい，結果としてえられるベクトルは要素数 (ベクトル長) が一定ではなく，処理をへるたびに要素数が変化するばあいがおおい．要素数が減少するばあいにはもとのベクトルにわりあてられた領域をそのまま使用することが可能だが，要素数が増大するばあいにはそのベクトルの領域の再わりあてが不可欠である．しかし，要素数が増大するたびにベクトル全体をつくりかえると，つぎのような問題点が生じる．第 1 に，ベクトル再わりあてのたびにもとのベクトルの要素をあたらしいベクトルにすべて複写しなければならないため，実行時間がふえる．第 2 に，ベクトル再わりあてによって記憶消費量がふえる．マルチ・ベクトルを使用すれば，これらのオーバヘッドをさけることができる．図 9.2 においても，複数のベクトルの複写をともなう併合によって生じるベクトル再生成オーバヘッドをへらすためにマルチ・ベクトルを生成しているとかんがえることができる．

9.4.2 バックトラック・並列処理の処理単位

マルチ・ベクトルはまた，並列バックトラックや並列処理における処理単位をきめるという点でも重要である．

第 1 に並列バックトラックについてのべる．並列バックトラック技法については 6.5 節で説明したが，これは解探索のベクトル処理において使用される，くみあわせ爆発をふせぐことをおもな目的とするプログラミング技法である．

図 9.6 にしめすように，並列バックトラック技法においてはマルチ・ベクトルを構成するひとつの部分ベクトルが処理単位となる．すなわち，並列バックトラック技法においては，ひとつの部分ベクトルの要素である部分解の数が膨大になり処理単位をわけたほうがよい状態になると，部分ベクトルの分割がおこなわれる．そして，それ以降の計算は部分ベクトルごとに逐次におこなわれる．図 9.6 においては，マルチ・ベクトル *m1* を構成する唯一の部分ベクトル *p1* が *p21* と *p22* とに分割され，それらを要素とするマル

チ・ベクトル *m2* がつくられている．分割の結果としてえられたマルチ・ベクトルを構成するひとつの部分ベクトルに関する計算から解がえられなかったとき，またはそこから解がえられたがさらに解をもとめるばあいには，バックトラックが発生してそのマルチ・ベクトルのつぎの部分ベクトルの処理がおこなわれる．

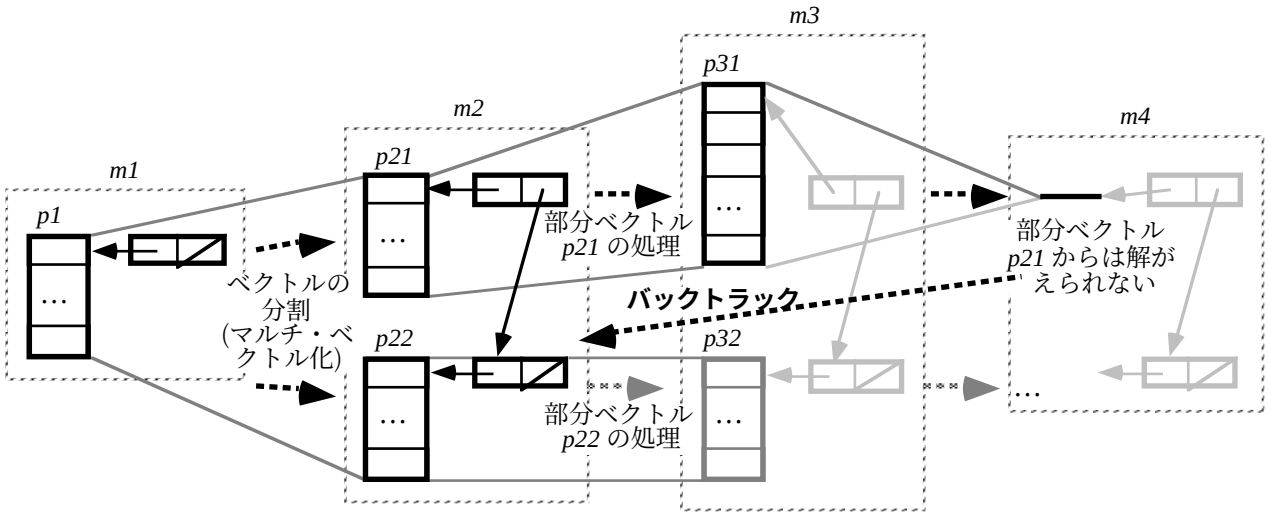


図 9.6　並列バックトラック技法におけるマルチ・ベクトルの使用法

ひとつの部分ベクトルに関する計算でえられた解をふくむベクトルはもとの部分ベクトルに対応してつくられ，あらたなマルチ・ベクトルの部分ベクトルとなる．図 9.6 においてはマルチ・ベクトル *m2* の最初の部分ベクトル *p21* を入力してベクトル *p31* がつくられ，バックトラックのあとで部分ベクトル *p22* を入力してベクトル *p32* がつくられる．そして, *p31* および *p32* を要素とするマルチ・ベクトル *m3* がつくられる[注4]. マルチ・ベクトル *m4* も同様にしてつくられる．

上記のことから，並列バックトラック技法においてはマルチ・ベクトルが重要な役割をはたしているということができる．なぜなら，すべての部分解がひとつのベクトルにふくまれているばあいはこのようにマクロに処理をわけることは困難であるから複数のベクトルが必要であり，さらにそれらのベクトルをまとめてあつかうにはそれらを連結してマルチ・ベクトルのかたちにしておくのが適当だとかんがえられるからである．

第 2 に，素数生成などのストリーム並列処理のベクトル計算機による実行についてのべる．このばあいには，マルチ・ベクトルが並列処理の処理単位として使用されうる．並列論理型言語による素数生成プログラムの動作は図 9.4 にしめしたが，第 7 章ではこのようなプログラムをベクトル処理するために，部分ベクトルを単位とするスケジュリングをおこなっている．

すなわち，整数列をふくむマルチ・ベクトルに対するつぎのような素数の倍数のフィ

[注4] *p31* がつくられた時点ではマルチ・ベクトル *m3* の尾部は未定である．

ルタリング処理のくりかえしの結果として素数 (素数を要素とするマルチ・ベクトル) が生成される．ひとつの部分ベクトルに関する素数の倍数のフィルタリング処理がまとめて一連のベクトル命令で実行される．この方法は，並列論理型言語のスケジュリング法としては**ふかさ優先スケジュリング** (depth-first scheduling) に対応する．しかし，ひとつのマルチ・ベクトルを構成することなる部分ベクトルに関するフィルタリング処理は，**幅優先スケジュリング** (breadth-first scheduling) によっておこなわれる．すなわち，ひとつのマルチ・ベクトルを構成する部分ベクトルの処理がおわると，そのマルチ・ベクトルの後続の部分ベクトルの処理はもっともひくい優先度をあたえられ，フィルタリングの過程で生じる他のマルチ・ベクトルを構成する部分ベクトルの処理が優先的に実行される．したがって，全体としてのスケジュリング戦略は，並列論理型言語の効率的なスケジュリング方法である**有界ふかさ優先スケジュリング** (bounded depth-first scheduling) に相当するやりかたでおこなわれる．

9.4.3 ベクトル・パイプラインの有効活用

パイプライン型ベクトル計算機は，ひとつのベクトルの各要素を短ピッチのパイプラインで計算する機構をもっている．このようなパイプラインをベクトル・パイプラインとよぶ．通常のベクトル計算機には，(疑似) マルチ・ベクトルの操作を支援するハードウェアは存在せず，マルチ・ベクトルの全部分ベクトルを一度にあつかえる命令は存在しない．したがって，複数の (部分) ベクトルのすべての要素に同一の演算をほどこすばあいには，ベクトルごとに 1 個の命令を実行する必要がある．このばあい，ベクトルの平均要素数が減少するとベクトル・パイプラインの使用効率は低下し，性能の低下をまねく．

しかし，もしマルチ・ベクトル支援機構をベクトル計算機に装備して，マルチ・ベクトル全体を処理するベクトル命令をつくれば，**図** 9.7 にしめすように，このような性能低下を部分的にふせぐことが可能になる．なぜなら，マルチ・ベクトルのデータ構造にあわせて，ハードウェアで最適なベクトル・パイプラインのスケジュリングをおこなうとともに，むだな処理をはぶくことができるからである．このような機構が存在するばあいにはマルチ・ベクトルを使用すれば，ひとつの命令であつかうことができないばらばらのベクトルを使用するのにくらべて，高速にベクトル処理をおこなうことが可能になる[注5]．

[注5] ただし，マルチ・ベクトル支援機構以外にこれに匹敵する高速化をする機構をつくることがまったくできないわけではない．また，このような機構においては，マルチ・ベクトルをつかわずばらばらのベクトルのままでも高速処理できる可能性がある．

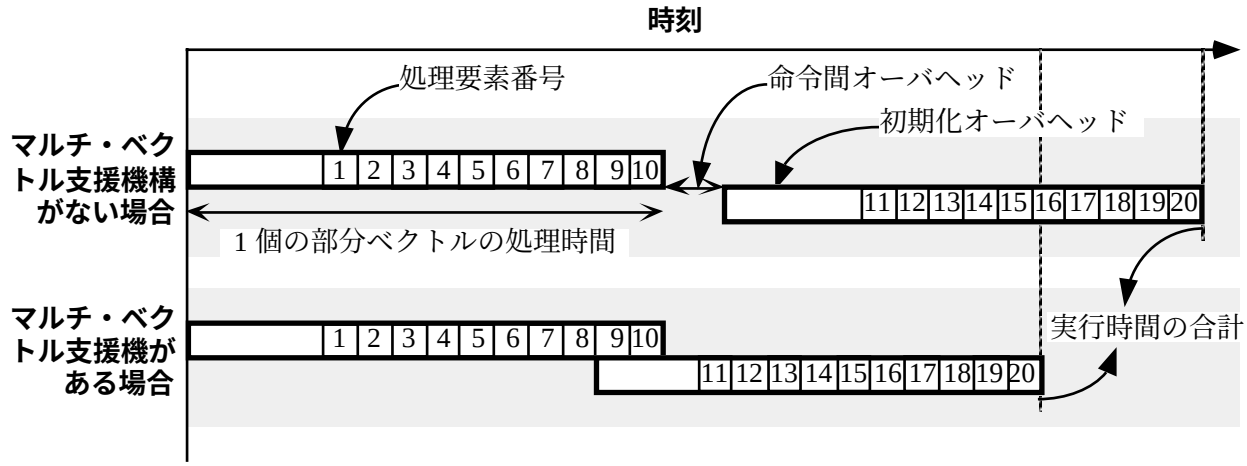


図 9.7　マルチ・ベクトル支援ハードウェアの効果

図 9.1 (a) や図 9.1 (b) のようなマルチ・ベクトルを 1 命令であつかえるベクトル命令をもつ計算機はまだ存在しないとかんがえられるが，M-680H IDP は図 9.5 でしめしたような疑似マルチ・ベクトル (マルチ・ベクトルに類似したデータ構造) に関するマージ処理を支援する機構をもっている．M-680H IDP はこの機構があるためにマージ・ソートを高速に実行することができる．マージ・ソートのベクトル処理においては，その実行開始時の疑似マルチ・ベクトルを構成する各部分ベクトルすなわちソートされたデータ列の要素数は 1 であり，処理がすすむにつれて部分ベクトルが併合されていく．そのため，処理がすすんだあとは部分ベクトルを単位とするベクトル処理をおこなっても十分な性能がえられるが，開始直後はベクトル長 (ベクトルの要素数) がみじかすぎるために，疑似マルチ・ベクトル全体を単位とするベクトル処理をおこなえる機構がなければ効率よく実行することができない.

また，2 次元配列が 1 次元配列をならべたものだとみなせば，これはマルチ・ベクトルに類似したデータ構造だということができるが，ベクトル計算機 ASC [Ramamoorthy 77] においては，2 次元配列を処理する命令によってマルチ・ベクトル命令と同様のやりかたでベクトル・パイプライン使用効率をあげている.

マージ・ソートのばあいにかぎらず解探索やストリーム処理においても，(疑似) マルチ・ベクトル処理を支援する機構があればベクトル再生成オーバヘッドをさらにへらすことが可能になる．なぜなら，部分ベクトルの併合回数をへらすことによって部分ベクトルの平均要素数が減少しても，ベクトル処理性能を維持することができるからである.

9.5 マルチ・ベクトルの操作法

マルチ・ベクトルに関するおもな操作は，部分ベクトルの分割と併合である．これらの操作についてのべる．

9.5.1 部分ベクトルの分割

マルチ・ベクトルを構成する部分ベクトル要素数がおおきくなりすぎると，たとえば 9.4.2 節でのべたくみあわせ爆発のような不都合が生じる．したがって，このようなばあいには部分ベクトルの分割をおこなうのがよい．部分ベクトル分割法としては，**図** 9.8 にしめすような複数の方法がかんがえられる．すなわち，要素数が一定数をこえた部分ベクトルだけを分割し，他の部分ベクトルはもとのままとする単純分割法 (a) と，各部分ベクトルの要素数のバランスを考慮して併合をともなう分割をおこなう方法 (b) とがかんがえられる．しかし，(b) の方法は併合によるオーバヘッドがあることをかんがえると，通常は (a) のような単純な方法がよいであろう．

上記の分割法の選択よりも重要なのは，分割の条件をきめることである．すなわち，単純分割法を採用するとして，部分ベクトル要素数が何個以上になったときに分割するか，また 1 個の部分ベクトルを何個に分割するかをきめることがより重要である．どのような分割条件がよいかはハードウェアや応用に依存するので一概にいえず，ベンチマーク・テストをおこなったうえで決定するのがのぞましい．しかし，一般には要素数がベクトル・レジスタ長の数倍程度すなわち 256 〜 2048 になったときに分割するようにし，必要以上に要素数が減少するのをさけるためには，部分解の増加がとくに急速でないかぎりは分割数は 2 とするのがよいであろう．

なお，部分ベクトル要素数の増大に対する対策としては，部分ベクトルの分割以外にも要素数の増大そのものの発生をなくす方法もある．並列バックトラック技法では，次節でのべる部分ベクトルの併合を意図的におこなわないかぎりは部分ベクトル要素数が増大することがない．それは，図 9.2 にしめしたように，部分解の数がふえるときにはもとの部分ベクトルと要素数がひとしいあらたな部分ベクトルを生成し，部分ベクトル要素数をふやすことはないからである．並列バックトラック技法においては，これにより部分ベクトルの分割と併合によるオーバヘッドをともにさけることが可能になっているということができる．

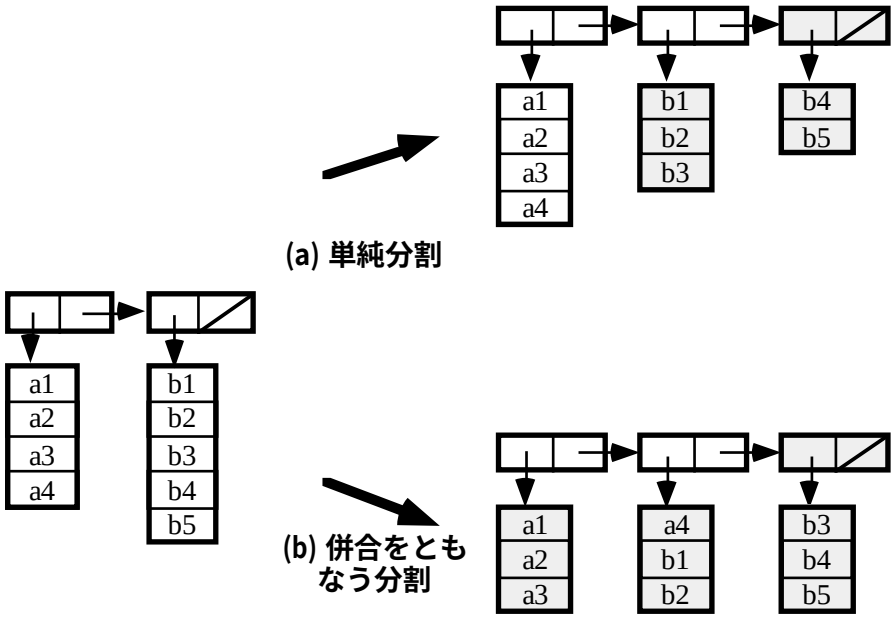


図 9.8　部分ベクトルの分割法

9.5.2 部分ベクトルの併合

金田らの方法によるエイト・クウィーン問題のような解探索のプログラムや並列論理型言語による素数生成プログラムにおいては，計算がすすむにつれて部分ベクトル要素数が減少する．この条件は素数生成のばあいになりたつことはあきらかだが，第 6 〜 7 章のベクトル処理方法においては，併合をおこなわないかぎりは部分ベクトル要素数は一定であるかまたは短縮するためになりたつ．このように部分ベクトルが短縮する一方なので，適当なところで**図** 9.9 に例示するような部分ベクトルの併合をおこなうのがよい．

部分ベクトルの併合法としては，**部分的併合** (図 9.9 a) と**完全な併合** (図 9.9 b) とがかんがえられる．完全な併合をおこなうと部分ベクトル要素数は最大になるので加速率は向上する．しかし単解探索のばあいには，部分ベクトル要素数をあまりおおきくすると不要な計算がふえるため，かえって効率が低下する (第 2 章参照)．また，部分ベクトル要素数がおおきくなると主記憶の消費がふえ，ばあいによってはデータが主記憶にはいりきらなくなって実行がつづけられなくなる．このようなばあいには，実行不能になるまえにあらかじめ部分的併合をおこなうようにするのがよい．

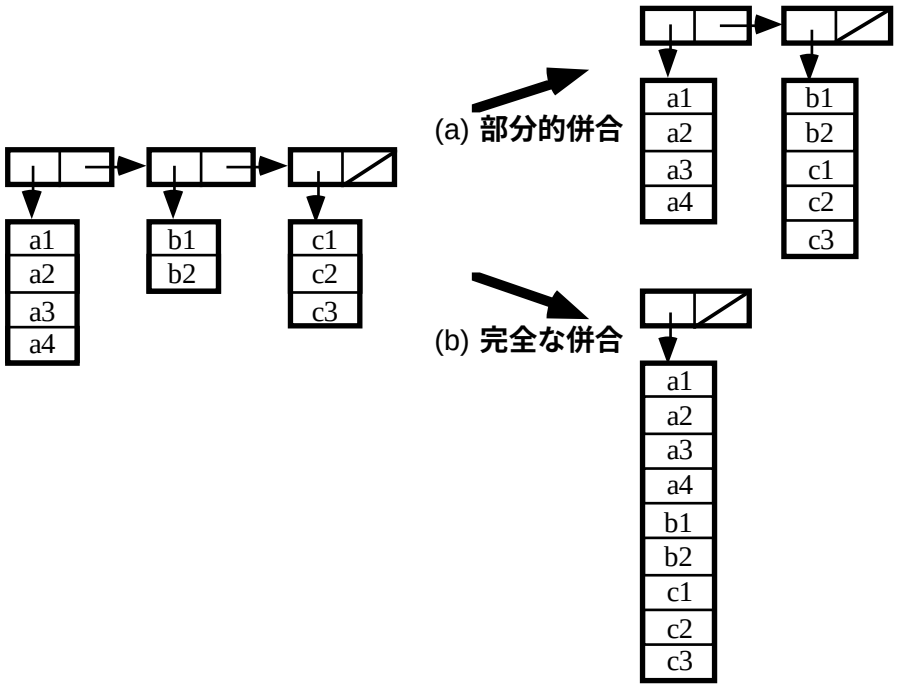


図 9.9　部分ベクトルの併合

部分ベクトルの併合の一般的な効果を議論できるだけの経験は蓄積していない．しかし，素数生成プログラムにおける部分ベクトルの併合の効果に関するかぎりは第 7 章で測定結果をしめしている．それによれば，併合をおこなわないばあいにくらべて，ばあいによっては 2 倍以上の高速化を実現している．しかし，これは併合をおこなわないばあいの部分ベクトルの平均要素数が 20 以下のばあいであり，もとの平均要素数が 61 のばあいには併合してもほとんど高速化されていない．

どのような条件のもとで併合をおこなったらよいかは，分割のばあいと同様にハードウェアや応用に依存する．しかし，上記の結果からは，部分ベクトル要素数が 50 程度以下になったら併合をおこなうのがよいとかんがえられる．

9.6 まとめ

9.2 〜 9.5 節における検討結果はつぎのようにまとめることができる．

❑ マルチ・ベクトルの応用

ベクトル記号処理，とくに並列バックトラック技法などによる解探索，素数生成のようなストリーム処理あるいはフィルタリング処理，マージ・ソートなどにおいてマルチ・ベクトルを使用することができる．

❑ マルチ・ベクトルの機能

マルチ・ベクトルは，解探索などの可変長ベクトル処理において問題となるベクトル再生成オーバヘッドを回避にやくだつ．またマルチ・ベクトルは，並列バックトラック技法におけるバックトラックやストリーム並列処理の処理単位としてはたらく．さらにマルチ・ベクトルは，ベクトル計算機にマルチ・ベクトル処理機構をもうけることによって，ベクトル・パイプラインの有効活用にやくだてることができる．

❑ マルチ・ベクトルの操作法

マルチ・ベクトルに対する重要な操作としては部分ベクトルの分割と部分ベクトルの併合とがあり，これらをおこなう条件を適切にきめることがベクトル処理性能をたかめるうえで重要である．

この研究の結果，ベクトル記号処理，とくに解探索，ストリーム処理，ソートなどにおいてマルチ・ベクトルを使用することができること，マルチ・ベクトルはベクトル再生成オーバヘッドを回避にやくだつこと，バックトラックや並列処理の処理単位としてはたらくこと，そしてベクトル・パイプラインの有効活用にやくだつことなどがわかった．これらの結果から，マルチ・ベクトルはリスト処理，データベース処理をはじめとするさまざまなベクトル記号処理において使用することができる非常に重要なデータ構造であると結論することができるだろう．

マルチ・ベクトルに関する今後の課題としては，この報告でしめした応用以外にもマルチ・ベクトルが有効であるかどうかをたしかめること，この報告でしめした以外の形式もふくめてどの形式のマルチ・ベクトルがより有効でありハードウェア支援に値するかをたしかめることなどがあげられる．また，ベクトル記号処理においてマルチ・ベクトル以外にも有用なデータ構造が存在するかどうか，それをどのようにつかうことができるかをしらべるのも今後の課題である．