

ベクトル処理の要素並列パイプライン方式  
による高速化とその実現方式に関する研究

河 辺 峻

ベクトル処理の要素並列パイプライン方式  
による高速化とその実現方式に関する研究

河辺 峻

# 目 次

第1章 序 論	1
1.1 本研究の背景	1
1.2 本研究の目的	4
1.3 本論文の構成	4
第2章 コンピュータの高速化方式におけるベクトル処理方式の重要性と 本研究の位置付け	6
2.1 コンピュータの高速化方式とベクトル処理方式	6
2.1.1 汎用コンピュータのアーキテクチャと問題点	6
2.1.2 内蔵ベクトル演算方式のアーキテクチャ	8
2.1.3 スーパーコンピュータのベクトル演算アーキテクチャ	10
2.2 ベクトル演算の種類とベクトル化	12
2.3 ベクトル命令の種類と命令数	14
2.4 リバモア・14 カーネルでのベクトル化状況	15
2.5 ベクトル・プロセッサの発展と本研究の位置付け	20
2.6 ベクトル・プロセッサの発展とコンパイラ技術との関連	23
第3章 ベクトル処理方式の分析と問題点	25
3.1 ベクトル処理方式の概念	25
3.2 S-810 におけるベクトル処理の実現方式	27
3.3 S-810 の課題と S-820 の開発技術	28
第4章 要素並列型演算パイプライン方式	30
4.1 S-820 におけるベクトル処理の実現方式	30
4.2 要素並列型演算パイプライン方式	32
4.3 要素並列型演算におけるベクトル・レジスタ構成	37
4.4 ベクトル演算制御の実現方式例	39
4.5 コンパイラによる高速処理方式	42
4.5.1 ベクトル命令が短い場合の高速化	42
4.5.2 プログラム変換による高速化	43
第5章 要素間に相互依存性のある処理の高速化方式	48
5.1 総和型演算の高速化方式	48

5.2 巡回型演算の高速化方式	52
5.3 コンパイラ等による高速処理方式	62
5.3.1 逐次加算演算の特殊処理	62
5.3.2 巡回型演算を含む高速化	62
第6章 ベクトル処理の分割起動方式と主記憶制御方式	64
6.1 ベクトル処理の分割起動方式	64
6.2 ベクトル処理の主記憶制御方式	68
6.2.1 S-820 における高速主記憶制御方式	68
6.2.2 セクション番号アサイン方式による高速化	70
6.2.3 バンク・グループ番号モディファイ方式による高速化	71
第7章 実現した方式の性能評価	74
7.1 要素並列型演算パイプライン方式の性能評価	74
7.2 総和型演算の性能評価	78
7.3 巡回型演算の性能評価	79
7.4 主記憶スループットの評価	80
7.5 総合性能評価	83
第8章 今後の高速処理方式に関する考察	86
8.1 はじめに	86
8.2 半導体技術の進歩と今後の動向	86
8.3 アーキテクチャの動向と今後の課題	88
8.4 今後の高性能コンピュータの動向	92
第9章 結 言	94
謝 辞	97
参考文献	99
執筆論文一覧	105

## 目 次

図 2.1 汎用コンピュータの処理の概念図	6
図 2.2 内蔵ベクトル演算方式におけるベクトル命令処理の概念図	8
図 2.3 スーパーコンピュータのベクトル演算処理の概念図	10
図 2.4 ベクトル演算におけるベクトル化率	12
図 2.5 実際のプログラムにおける演算内容	13
図 2.6 リバモア・14 カーネル ソースリスト	17
図 2.7 ベクトル・プロセッサの最大性能の動向	20
図 3.1 ベクトル演算の概念と高速化の理由	26
図 3.2 HITAC S-810 の論理構造概略図	27
図 4.1 HITAC S-820 の論理構造概略図	31
図 4.2 S-810 の演算パイプライン方式	33
図 4.3 要素並列型演算パイプライン方式	36
図 4.4 要素並列処理の例	38
図 4.5 S-820 のベクトル演算制御方式	39
図 4.6 ベクトル命令制御タイムチャート概略	41
図 4.7 $A(i)=B(i)+C(i)$ のコンパイラによる高速化方式	42
図 5.1 要素並列処理(パイプライン)	49
図 5.2 要素並列処理(後処理演算器)	51
図 5.3 演算ユニットの構成	53
図 5.4 逐次加算演算の高速化(VINC)	56
図 5.5 逐次加算演算の通常処理(VINC)	57
図 5.6 一次巡回演算の高速化(VITR)	60
図 5.7 一次巡回演算の通常処理(VITR)	61
図 6.1 ベクトル処理の分割起動方式	65
図 6.2 ベクトル処理の分割起動方式例	67
図 6.3 HITAC S-820 の主記憶制御構成図	68
図 6.4 逐次処理による主記憶制御	70
図 6.5 セクション番号アサイン方式による主記憶制御	70
図 6.6 セクション番号アサイン方式	71
図 6.7 バンク・グループ番号モディファイなしの主記憶アクセス	72



図 6.8 バンク・グループ番号モディファイありの主記憶アクセス .....	73
図 7.1 S-810 と S-820 の演算器およびロード・ストア回路利用率比較 .....	75
図 7.2 要素並列型演算バイプライン方式の方式評価 .....	76
図 7.3 ベクトル・ロード命令の主記憶スループット .....	80
図 7.4 ベクトル・ストア命令の主記憶スループット .....	81
図 8.1 半導体技術の進歩と DRAM の集積度の進歩 .....	87
図 8.2 Moore の法則 .....	87
図 8.3 Linpack 1000 x 1000 性能 .....	89
図 8.4 ベクトル処理と RISC 処理 .....	89
図 8.5 RISC の性能とベクトル処理性能 .....	91
図 8.6 TFOPLS へのアプローチ .....	91
図 8.7 高速計算処理の課題 .....	92

## 表 目 次

表 2.1 計算内容とベクトル処理可能性 .....	14
表 2.2 ベクトル命令の種類と命令数 .....	15
表 2.3 リバモア・14 カーネルでのベクトル化状況 .....	16
表 2.4 ベクトル・プロセッサの発展の概要 .....	21
表 2.5 ベクトル・プロセッサの発展とコンパイラ技術との関連 .....	23
 表 3.1 S-810 の課題と S-820 の開発技術 .....	 28
 表 5.1 後処理演算器で高速化を図ったベクトル命令の種類 .....	 50
表 5.2 専用制御演算器で高速化を図ったベクトル命令の種類 .....	52
 表 7.1 S-820/S-810 の演算器およびロード・ストア回路利用率 .....	 75
表 7.2 後処理演算器で高速化を図った内積演算の性能例 .....	78
表 7.3 専用制御演算器で高速化を図った演算の性能例 .....	79
表 7.4 S-820 の性能評価 (リバモアグループ) .....	83

## 第1章 序論

### 1.1 本研究の背景

科学技術の進歩に伴い、多くの分野で大規模な計算が必要になってきている。歴史的には原子力、航空工学、気象予報などの大規模計算を対象に、一般にスーパーコンピュータと呼ばれる科学技術計算専用機が、主として米国で開発され、使用されてきた。スーパーコンピュータという言葉の明確な定義はないようであるが、一般には、その時代の汎用コンピュータの性能に比べて、科学技術計算が格段に速いコンピュータをスーパーコンピュータと呼んでいる。

スーパーコンピュータの特徴としては「並列処理」や「ベクトル処理」がその高速処理技術としてあげられる。特に1970年代前半から商用化された「ベクトル処理」を行うスーパーコンピュータは大きく発展を遂げ、現在商用化され実用に供されているスーパーコンピュータは、ほぼすべてといってよいほどこの「ベクトル処理」と呼ばれる高速演算方式を採用し、これによって高い性能を実現している。このため、スーパーコンピュータは「ベクトル・プロセッサ」または「ベクトル・コンピュータ」とも呼ばれている。

ベクトル・プロセッサが商用機として初めて登場したのは、1970年代前半で TI(Texas Instruments)社の ASC(1972年)<sup>1)</sup>、CDC(Control Data Corporation)社の STAR-100(1973年)<sup>2)</sup>が有名である。ASCはadvanced scientific computer、STARはstring arrayの略で、いずれも性能は50MFLOPS(ASCがマシンサイクル80nsピッチで4個の浮動小数点演算器結果が得られ、STAR-100がマシンサイクル40nsピッチで2個の浮動小数点演算器結果が得られた)で、主記憶上のベクトル・データを直接パイプライン演算器に入力して演算する方式を採用していた。

この第1世代のベクトル・プロセッサの課題は、ベクトル演算性能向上率の向上であった。すなわち、ベクトル長(ベクトル演算要素数)が数百程度と長くなければ十分な性能向上を得ることができなかった。その理由は、すべてのベクトル・データを主記憶から読み出して演算し、結果を格納しなければならない、これに時間を要していたからである。また当時はコンパイラ技術が未熟で「自動ベクトル化」のコンパイラがなかったため、ユーザが自分でサブルーチン・コールの形でベクトル演算を利用しなければならなかった為、高価格とあいまって、全世界でも十数台が利用されたに過ぎなかった。

1976年に、CDCから独立したSeymore Crayが設立したCray Research社からCRAY-1が出荷された<sup>3)</sup>。性能は160MFLOPSと当時としては画期的な高性能であり、第2世代



の幕開けにふさわしいマシンであった。これが評価され、CRAY-1が商用スーパーコンピュータとして初めて成功したマシンとなった。

高性能実現の背景として、ベクトル・レジスタおよびチェイニングと呼ばれる新しい方式の導入があげられる。当時は高速半導体 RAM 開発の黎明期であったが、初めて超高速の Bipolar RAM をベクトル・レジスタに適用し、ベクトル演算途中の結果を毎回主記憶に戻す必要がなくなった。また主記憶にもアクセス時間 25ns の Bipolar RAM を使用することにより、短いベクトル長でも高い演算性能向上率が得られるようになった。

さらに、チェイニングによって加算と乗算の並列実行が可能となり、実効性能が格段に向上した。これに加えて、4 ゲートの IC を極めて高密度に実装することにより、12.5ns のマシン・サイクルが実現され（当時の汎用コンピュータのマシン・サイクルは 70~80ns であり、いかに高速であったかがわかる）、160MFLOPS という高性能が達成された。

これに対して、CDC は STAR-100 をベースにこれを発展させた CYBER 203/205 を開発した<sup>14)</sup>。

一方、国内では富士通が FACOM/230-75AP (array processor)を開発し、1977 年に出荷した<sup>15)</sup>。最大性能は 22MFLOPS で、アーキテクチャとしては STAR-100 と同様、主記憶のベクトル・データを直接演算する方式であった。

また、日立製作所では汎用コンピュータ M シリーズに付加するベクトル・コンピュータとして IAP(integrated array processor: 内蔵アレイプロセッサ)を開発し<sup>16) 17)</sup>、1978 年から出荷した。オペレーティングシステムを改造することなくマルチプログラミング環境を実現するために、プログラム実行の中断・再開時に退避・回復が必要なベクトル・レジスタなどを新設することはやめることにしたため、アーキテクチャとしては、これも主記憶のベクトル・データを直接演算する方式であった。しかし「自動ベクトル化」コンパイラの研究・開発を進め自動ベクトル化コンパイラの商用化に成功し<sup>18)</sup>、その後の本格的なスーパーコンピュータの発展に大きく寄与した。

この頃になると、利用分野が拡大し、構造計算、分子科学、核融合、半導体、資源探査、環境分析、エネルギー給配、経済分析などいろいろな分野で高速計算のニーズが高まってきた<sup>19)</sup>。

分野が多岐にわたるにつれ、計算を必要とする問題も多様化している。物理の方程式を高精度に解くこと、大量の数値データに一定の計算処理を行うこと、現象の時間的経過をシミュレーションすること、物理実験や工学実験に代えてシミュレーションすることな

ど、スーパーコンピュータの使い方新しい道が開かれつつある。そして、いずれの問題にも、膨大な数値データを処理する巨大な計算パワーが必要になってきている。

このような状況で国産のスーパーコンピュータが誕生した。1983年に日立製作所が S-810 シリーズ (最大性能 630MFLOPS)<sup>1-10)</sup>、富士通が VP シリーズ (最大性能 500MFLOPS)<sup>1-12)</sup>、1985年に NEC が SX-1/2 (最大性能 1.3GFLOPS)<sup>1-14)</sup> を出荷した。これらが第3世代のスーパーコンピュータと呼ばれる。

その後第4世代のスーパーコンピュータとして、1988年に日立製作所が S-820 シリーズ (最大性能 3GFLOPS)<sup>1-15)</sup>、富士通が 1990年に VP-2000 シリーズ (最大性能 5GFLOPS)<sup>1-16)</sup>、同じく NEC が SX-3 (最大性能 22GFLOPS(4CPU 構成))<sup>1-17)</sup> を出荷し、それぞれ出荷当時において世界最高レベルの性能を実現してきた。1CPU のベクトル・プロセッサの性能で見ると、国産3社が激しい競争を通じて世界をリードしてきたといっても過言ではない (本論文のテーマもこれらの研究開発によって得られたものを主としている)。

なお、これまで国産のマシンはシングルプロセッサ方式を採用してきたが、SX-3において初めてマルチプロセッサ(4CPU)を実現した。1992年には NEC が SX-3 を強化した R シリーズ (最大性能 25.6GFLOPS(4CPU 構成)) を、日立製作所が S-3000 シリーズ (最大性能は S-3000 の 32GFLOPS(4CPU 構成))<sup>1-18)</sup> を発表した。

これに対し、海外では米国の Cray Research 社が主として世界市場をリードした。Cray Research 社は 1983年に CRAY X-MP シリーズ (最大性能 420MFLOPS(2CPU 構成))<sup>1-19)</sup> を出荷し、世界で初めて 2CPU のマルチプロセッサを実現した(その後 4CPU 構成に拡張した)。また 1984年に CRAY-2(最大性能 2GFLOPS(4CPU 構成))<sup>1-20)</sup>、1989年に CRAY Y-MP シリーズ(最大性能 2.7GFLOPS(8CPU 構成))<sup>1-21)</sup> を出荷した。CRAY Y-MP は最大 8CPU のマルチプロセッサ構成を可能とした。

一方、CDC はスーパーコンピュータ部門を分離独立させて ETA systems 社を設立して ETA10 を開発し、1988年に出荷した。このプロセッサは CMOS LSI と液体窒素冷却を組み合わせた実装技術を使用し<sup>1-22)</sup>、8CPU 構成で最大性能 10GFLOPS を実現している。しかし、期待したようには発展せず、1990年にこの分野から撤退した。

1989年には CRAY-2 の後継機である CRAY-3 の開発を担当していた Seymour Cray が Cray Research 社から独立して Cray Computer 社を設立した。Cray Computer 社は CRAY-3 のプロトタイプを開発したのみで、1996年に倒産した。

Cray Research 社は 1992 年に CRAY Y-MP C90 シリーズ(最大性能 24GFLOPS(16CPU 構成)) を出荷し、1996 年には CRAY T90 シリーズ(最大性能 64GFLOPS(32CPU 構成)) を出荷した。

## 1.2 本研究の目的

高速なコンピュータを実現するためには、

- ・アーキテクチャやプロセッサの論理方式上の工夫。
- ・半導体素子やその実装上の工夫。

に大きく分けられる。本研究は、ベクトル・プロセッサにおけるシングル・プロセッサの高速化に関し、主として前者のアーキテクチャやプロセッサの論理方式上の研究を対象としている。

このため本研究は、まず科学技術計算における高速化技術であるベクトル演算処理方式の効果について論じる。ベクトル演算処理アーキテクチャとしては、

- ・主記憶上のデータを直接演算器に入力して演算する方式
- ・主記憶上のデータをベクトル・レジスタで受けて演算する方式

の2つが考えられる。これらと比較し、現在の「ベクトル・レジスタ演算方式」が採用された理由を説明する。

次にベクトル演算処理方式が有効に働くための、ベクトル演算の種類とベクトル化率およびそれを実現するアーキテクチャであるベクトル命令の種類と命令数について論じる。

さらにそのアーキテクチャを高速に処理するための、プロセッサの論理方式について論じる。特に、ベクトル・プロセッサにおけるシングル・プロセッサの高速化に関し、「要素並列型演算パイプライン方式」技術を開発し、実際の製品に適用 (HITAC S-820) した結果を示す。

## 1.3 本論文の構成

本論文は、以上のような背景と目的を持って、筆者が 1974 年以来研究開発を担当してきたベクトル演算処理方式の成果を纏めたものである。当初は、汎用コンピュータにベクトル演算処理を付加する、内蔵アレイプロセッサ方式 (これは M-180 IAP, M-200H IAP, M-280H IAP として製品化された) の検討から始まり、その後、本格的なベクトル・プロセッサである HITAC S-810 およびその後継機種である HITAC S-820 の検討へと進んだ。本論文の構成および各章の概要は、次のようである。

第1章に続き、第2章ではコンピュータの高速化方式におけるベクトル処理方式の重要性およびベクトル演算の種類とベクトル化、ベクトル命令の種類と命令数について述べ、リバモア・14カーネルでのベクトル化状況を評価する。さらにベクトル・プロセッサの発展と本研究の位置付け、およびコンパイラ技術との関連についても述べる。

第3章では、ベクトル処理方式の分析と問題点について述べる。ベクトル処理方式の概念と高速処理の原理についてまず述べ、S-810において実現したベクトル処理方式について述べる。さらに、S-810のベクトル処理方式の分析を行い、課題と後継機種であるS-820の開発技術を明らかにする。

第4章では、まずS-820において実現したベクトル処理方式について述べる。さらにS-820の特長技術の一つである、要素並列型演算パイプライン方式について詳細を述べる。さらに第5章において、要素間に相互依存性のある処理の高速化方式について述べ、第6章において、ベクトル処理の分割起動方式およびベクトル処理の主記憶制御方式について述べる。いずれもS-820において開発し実現した技術である。

第7章では、本研究で開発し、実現した各種の高速処理方式の性能評価を行う。特に、要素並列型演算パイプライン方式の評価および要素間に相互依存性のある処理として総和型演算と巡回型演算の性能評価について詳細を述べる。また合わせて主記憶スループットの評価についても述べる。

第8章において、今後の高性能コンピュータの高速処理方式に関する考察について触れ、第9章に結言を述べ、本論文を纏める。



## 第2章 コンピュータの高速化方式における ベクトル処理方式の重要性と本研究の位置付け

### 2.1 コンピュータの高速化方式とベクトル処理方式

#### 2.1.1 汎用コンピュータのアーキテクチャと問題点

現在の汎用コンピュータ（CISC 型）のアーキテクチャの基本的な考え方は、2つのオペランドの演算を指示する命令を次々に実行して処理を進めて行くことである。図 2.1 に汎用コンピュータの処理の概念図を示す。

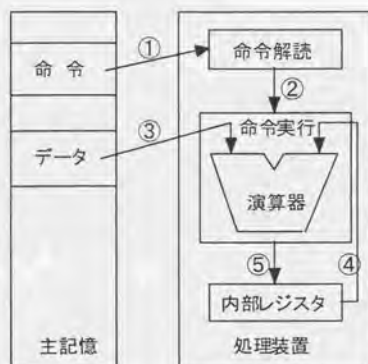


図 2.1 汎用コンピュータの処理の概念図

この図において動作を指示する命令は主記憶にある。命令の実行は処理装置による命令の読み出しとこれに続く命令解釈によってその実行が開始される。通常の命令は2つのオペランドを指示し、この図では一方のデータは主記憶、他方は処理装置内の高速の内部レジスタから読み出される。2つのオペランドは演算器で処理され内部レジスタへ戻される。

これが代表的な命令の処理形態であり、2つのオペランドが主記憶あるいは内部レジスタにあるかにより、命令の処理タイプが、

- ・レジスタ — レジスタ
- ・レジスタ — 主記憶（図 2.1 の例）



・主記憶 — 主記憶  
に分類される。

次に科学技術計算における汎用コンピュータの処理とその問題点について考える。科学技術計算のプログラムは主として FORTRAN 言語で記述されているが、処理時間の多くの部分は DO ループの処理である。例えば、簡単のために、

```
DO 10 I = 1, N
10 A(I) = B(I) + C(I)          【例1】
```

を例にとって考える。汎用コンピュータの命令ではこの DO ループは FORTRAN コンパイラによって次のような命令に展開される。

```
LD 0, C(I)
AD 0, B(I)
STD 0, A(I)
AR      (オペランドアドレス更新)
BCT     (ループ回数制御: N 回ループ)
```

従って、この DO ループを処理するためには、主記憶より

```
2 x N のオペランドの読み出し
1 x N のオペランドの書き込み
5 x N の命令の読み出し
```

を必要とする。

この例においてまず着目すべき点は、オペランドの転送量よりも命令の転送量の方が多いという点である。DO ループという決まった手順の繰り返しに、5 N 回もの命令の読み出しを行うのはいかにももったいない。従って、コンピュータ内部の実現方式（論理方式技術）では、命令が主記憶に連続して格納されていること、および同じ命令をたびたび頻度高く転送することなどの性質を利用して、

- ・一度に複数個の命令を転送する。
- ・処理装置内に高速のキャッシュ・メモリを置き、たびたび用いられる主記憶の写しを持っておく。

などの処理を行うことにより、主記憶と処理装置との間の命令の転送を実効的に減らしている。

これはアーキテクチャが要求する一見無駄な処理を、論理方式技術がカバーしている一例である。しかしながら処理装置の内部では 5N 回の命令の解読と実行が行われており、オペランドのアドレスの計算も各命令の解読の後に毎回行われる。

これらの反省からベクトル命令の概念が生まれた。ベクトル命令では、決まった手順の繰り返しで処理するオペランドの転送と演算を一つの命令で済ませることができる。先程の DO ループの例では、

#### VEAD (Vector Element-wise Add Double) 命令

という一つのベクトル命令を処理すれば良く、処理装置は 3 N 回のオペランドの転送と、1 N 回の演算（加算）に専心することができる。

#### 2.1.2 内蔵ベクトル演算方式のアーキテクチャ

内蔵ベクトル演算方式とは、科学技術計算能力を高めるために、日立製作所が HITAC M-180, M-200H, M-280H などの汎用大型コンピュータ上に、IAP(Integrated Array Processor)機能として実用化したものである。図 2.2 に内蔵ベクトル演算方式におけるベクトル命令処理の概念を示す。

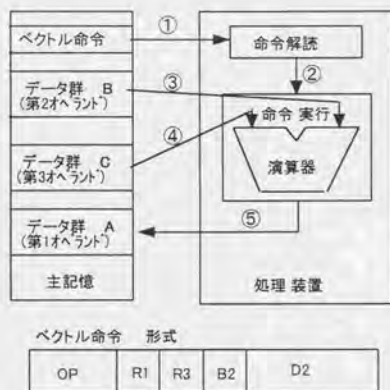


図 2.2 内蔵ベクトル演算方式におけるベクトル命令処理の概念図

ベクトル命令の仕様は、図に示すように第 2 オペランドと第 3 オペランドのデータ群の間でエレメント対応に演算を行い、結果を第 1 オペランドのデータ群に格納する。このよ

ベクトル命令の仕様は、図に示すように第2オペランドと第3オペランドのデータ群の間でエレメント対応に演算を行い、結果を第1オペランドのデータ群に格納する。このようにデータ群をひとまとめに定義することにより、処理装置は一括して演算を行い高速性能を得ている。

この方式の一つの特徴は、すべてのベクトル命令のオペランドが主記憶上に存在し、内部レジスタを用いていないという点である。これにより、ベクトル命令処理中に、処理を中断して割り込みを受け付けることが可能になる。処理されたエレメント数は、ベクトル命令の R1 フィールドで指定された汎用レジスタに格納され、割り込み処理後、再びこのベクトル命令が残りのエレメントの処理に戻る。こうすることにより、オペランドが主記憶上に存在しないとき（仮想記憶においてマッピング・フォールトが発生したとき）も処理するエレメント数のところで、割り込みが発生し、オペランドが主記憶上にページ・インした後、処理が再開される。

このように、汎用コンピュータ並みのきめ細かな割り込み制御を可能にすることにより、既存のオペレーティング・システムとの共存を実現し、ベクトル命令方式の用途を拡大することができた。

この方式の一つの問題点は、主記憶上のオペランドの参照回数が増大することがある点である。例えば、次の FORTRAN の DO ループの例を考えてみる。

```
DO 20 I = 1, N
20 A(I) = B(I) + C(I) + D(I)          【例2】
```

汎用コンピュータの命令ではこの DO ループは次のような命令に展開される。

```
LD    0, D(I)
AD    0, C(I)
AD    0, B(I)
STD   0, A(I)
AR                      (オペランドアドレス更新)
BCT   (ループ回数制御：N 回ループ)
```

従って、この DO ループを処理するためには、主記憶より

3 x N のオペランドの読み出し  
1 x N のオペランドの書き込み

6 x N の命令の読み出し

を必要とする。一方ベクトル命令方式では、次の2命令に展開される。

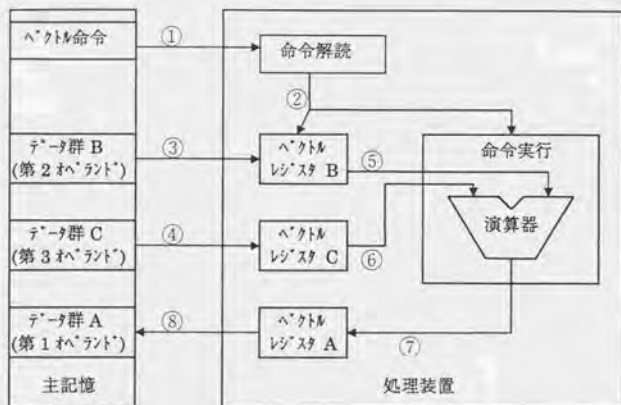
VEAD W < C + D

VEAD A < B + W

すなわち、ベクトル命令の読み出しは2回で済むが、オペランドの主記憶に対する参照は、本来の4N回に対して6N回に増大している。このため、演算の項数が増えるとオペランドの主記憶に対する参照が増大するため性能の向上率が低下する傾向になる。

### 2.1.3 スーパーコンピュータのベクトル演算アーキテクチャ

性能を最優先させたアーキテクチャがスーパーコンピュータのベクトル演算アーキテクチャである。これは図2.3に示すように、数10~100KBのベクトル・レジスタを処理装置内部に設けて主記憶とのデータ転送を大幅に減らしている。



ベクトル命令形式

OP	R1	R2	R3
----	----	----	----

図2.3 スーパーコンピュータのベクトル演算処理の概念図

図 2.3 の例は図 2.2 の例と同じく、【例 1】を例にしているので、両者に差異はない。むしろ図 2.3 の方がベクトル・レジスタを経由する分だけ複雑になっている。ところが、【例 2】で示す DO ループの場合は、オペランドの主記憶に対する参照は、内蔵ベクトル演算方式では 6N 回であったが、スーパーコンピュータのアーキテクチャでは本来の 4N 回に減少する。これは演算の中間結果をベクトル・レジスタには格納するが、主記憶には格納しないためである。



現在の代表的なスーパーコンピュータはすべてこの考え方をベースにして設計されている。このようにすれば、主記憶とのデータ転送量は必要最低限で済むことになり、性能上の隘路になり易い主記憶とのデータ転送はアーキテクチャ技術により解決される。

スーパーコンピュータの処理装置の設計は、主記憶とのデータ転送能力が最大限に発揮できるように行われる。このため処理装置内部の実現方式である方式論理技術では、

- ・並列パイプライン演算方式
- ・複数命令の並列演算実行方式

などを用いて複数のベクトル命令の演算、実行が複数のパイプライン化された演算器を用いて行われる。

この結果、コンピュータ内部の処理は必然的にデータフロー的に実行され、性能は主記憶とのデータ転送能力に大きく依存することになる。

## 2.2 ベクトル演算の種類とベクトル化

ベクトル演算ではFORTRANプログラムの最内側のDOループ内に含まれる演算が対象となる。ここには多種多様な演算が存在するが、これらをいかにベクトル演算させるかでベクトル化率が決まる。

ベクトル化率とは、図2.4に示すように、スカラ命令のみで実行した時間のうち、ベクトル命令で処理できる部分の比率である。この部分を増やすことが、ベクトル演算方式では極めて重要である。

$$T_v = ((1-\alpha) + \alpha/V) T_s$$

$\alpha$  : ベクトル化率

$V$  : ベクトル演算性能向上率

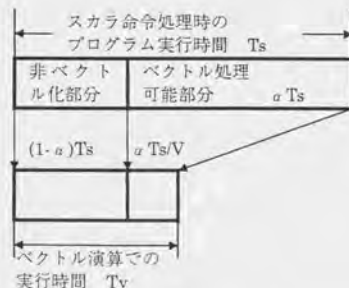


図2.4 ベクトル演算におけるベクトル化率

実際のプログラムにおける演算内容を調べた例を図 2.5 に示す。これは Fortran で記述したプログラムをすべて M シリーズの命令セットに展開し、動的実行ステップ数（動的命令ステップ数）の合計を 100 としたとき、どのような演算を何%実行したかを分析したものである。計算センタにおける比較的小規模な科学技術計算と、これとは別に、あるユーザの大規模な科学技術計算を取り上げている。

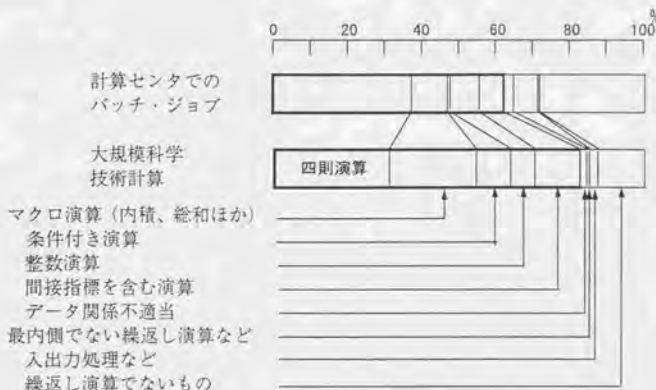


図 2.5 実際のプログラムにおける演算内容

単純な四則演算や内積、総和、3 項の積和などのマクロ演算をベクトル命令に展開しただけでは、計算センタのジョブでは 48%、大規模な科学技術計算の例では 55%をベクトル処理するにとどまることを示している。

ベクトル処理部分の比率を上げるには、さらに条件付き演算 (IF 文を含む DO ループ)、整数演算、間接指標ベクトル (リスト・ベクトル) 演算などをベクトル命令に展開し、ベクトル処理に持っていくかねばならない。

そこで表 2.1 に示すように、計算構成要素の広範囲にわたってベクトル命令に展開し、ベクトル処理ができるようにしなければならない。

表 2.1 計算内容とベクトル処理可能性

	構 成 要 素	Fortran 例	M-200H IAP	M-280H IAP	S-810 S-820
1	四則演算	$A(I) = B(I) + C(I)$	○	○	○
2	内積・総和・3項積和	$S = S + A(I) \times B(I)$	○	○	○
3	条件付き演算	$IF(A(I).EQ. B(I))$ $C(I) = D(I)$	×	○	○
4	整数・論理演算	$N(I) = L(I) + K(I)$	×	×	○
5	間接指標ベクトル (リスト・ベクトル)	$K = N(I)$ $A(K) = B(K) + C(K)$	×	×	○
6	集約・分散	$A(I) = B(L(I))$	×	×	○
7	関数(SIN, EXP, ...)	$X(I) = EXP(Y(I))$	○	○	○
8	参照関係不適	$A(I) = A(I-4) + B(I)$	×	×	×
9	DO ループ以外		×	×	×

この表には M-200H IAP および M-280H IAP の場合のベクトル処理可能な範囲とを対比させて示した。S-810/S-820 では IAP でのベクトル処理可能な範囲に加えて、スパース行列の処理によくでくる間接指標ベクトルに対する演算、整数・論理演算、関数などをベクトル命令に展開することができる。

これらにより、S-810/S-820 では計算センタのパッチジョブのような比較的規模の小さな計算では約 60%、大規模な科学技術計算では 80% 以上がベクトル命令に展開することが可能になった。

### 2.3 ベクトル命令の種類と命令数

ベクトル処理の範囲の拡大を図るために、表 2.2 に示すようなベクトル命令の種類と命令数を用意した。S-810/S-820 では 83 種類のベクトル命令をサポートした。

表 2.2 ベクトル命令の種類と命令数

	分 類	M-200H IAP	M-280H IAP	S-810 S-820
1	ロード・ストア	0	0	18
2	転送	4	4	7
3	四則演算	8	8	11
4	マクロ演算	14	20	10
5	フォーマット変換	2	2	2
6	比較	0	12	12
7	検索	0	0	7
8	論理演算	0	0	7
9	マスク演算	0	0	7
10	制御	0	0	2
	命令数合計	28	46	83

#### 2.4 リバモア・14 カーネルでのベクトル化状況

リバモア・14 カーネルは米国の Lawrence Livermore 国立研究所がベクトル・プロセッサの評価用に提案したもので、ベクトル・プロセッサの実効的な性能指標を得るためのベンチマークとして広く利用されている。典型的な科学技術計算プログラムの中から主要な DO ループを 14 個抽出し、各ループごとの性能を評価するものである。図 2.6 に 14 カーネルのプログラムリストを示す。

このリバモア・14 カーネルでのベクトル化状況を表 2.3 に示す。

表 2.3 リバモア・14 カーネルでのベクトル化状況

カーネル 番号	カーネルの内容	S-810	M-280	ベクトル化可能要因
		S-820	IAP	
1	流体	○	○	
2	MLR 内積	○	○	
3	内積	○	○	
4	帯型連立一次方程式	○	×	コンパイラ能力の向上
5	三角化消去 (下三角)	○	×	コンパイラ能力の向上
6	三角化消去 (上三角)	○	×	コンパイラ能力の向上
7	状態方程式	○	○	
8	P.D.E.積分	○	×	短ベクトル性能の向上
9	整数予測	○	○	
10	差分予測	○	○	
11	総和	○	○	
12	差分	○	○	
13	二次元粒子推進	△	×	間接指標ベクトル参照 コンパイラ能力の向上
14	一次元粒子推進	△	×	間接指標ベクトル参照 コンパイラ能力の向上

注) ○:ベクトル化可能 △:部分ベクトル化可能 ×:ベクトル化不可能



NO.	プログラム名	リバモア・14カーネル ソースリスト
1	流体	DO 1 K=1,400 1 X(K)=Q+Y(K)*(R*Z(K+10)+T*Z(K+11))
2	内積 (MLR)	DO 2 K=1,996,5 2 Q=Q+Z(K )*X(K )+Z(K+1)*X(K+1) + Z(K+2)*X(K+2)+Z(K+3)*X(K+3) + Z(K+4)*X(K+4)+Z(K+5)*X(K+5)
3	内積	DO 3 k=1,1000 3 Q=Q+Z(K)*X(K)
4	帯型連立一次方程式	DO 4 J=30,870,5 X(L-1)=X(L-1)-X(LW)*Y(J) 4 LW=LW+1
5	三角化消去(下三角)	DO 5 I=2,998,3 X(I )=Z(I )*(Y(I )-X(I-1)) X(I+1)=Z(I+1)*(Y(I+1)-X(I )) 5 X(I+2)=Z(I+2)*(Y(I+2)-X(I+1))
6	三角化消去(上三角)	DO 6 J=3,999,3 I=1000-J+3 X(I )=X(I )-Z(I )*X(I+1) X(I-1)=X(I-1)-Z(I-1)*X(I ) 6 X(I-2)=X(I-2)-Z(I-2)*X(I-1)
7	状態方程式	DO 7 M=1,120 X(M)= U(M) +R*(U(M )+R*U(M )) + T*(U(M+3)+R*(U(M+2)+R*U(M+1)) + T*(U(M+6)+R*(U(M+5)+R*U(M+4)) 7 CONTINUE
8	P.D.E.積分	DO 8 KX=2,3 DO 8 KY=2,21 DU1=U1(KX,KY+1,NL1)-U1(KX,KY-1,NL1) DU2=U2(KX,KY+1,NL1)-U2(KX,KY-1,NL1) DU3=U3(KX,KY+1,NL1)-U3(KX,KY-1,NL1) U1(KX,KY,NL2)=U1(KX,KY,NL1)+A11*DU1+A12*DU2+A13*DU3+ 1 SIG*( U1(KX+1,KY,NL1)-2.* U1(KX,KY,NL1)+ U1(KX-1,KY,NL1)) U2(KX,KY,NL2)=U2(KX,KY,NL1)+A21*DU1+A22*DU2+A23*DU3+ 1 SIG*( U2(KX+1,KY,NL1)-2.* U2(KX,KY,NL1)+ U2(KX-1,KY,NL1))

		$U3(KX,KY,NL2)=U3(KX,KY,NL1)+A31*DU1+A32*DU2+A33*DU3+ \\ 1 \text{ SIG}(U3(KX+1,KY,NL1)-2.*U3(KX,KY,NL1)+U3(KX-1,KY,NL1))$
9	整数予測	8 CONTINUE DO 9 I=1,100 $PX(1,I)=BM28*PX(13,I)+BM27*PX(12,I)+BM26*PX(11,I)+ \\ + BM25*PX(10,I)+BM24*PX(9,I)+BM23*PX(8,I)+ \\ + BM22*PX(7,I)+CO*(PX(5,I)+PX(6,I))+PX(3,I)$
10	差分予測	9 CONTINUE DO 10 I=1,100 AR=CX(5,I) BR=AR-PX(5,I) PX(5,I)=AR CR=BR-PX(6,I) PX(6,I)=BR AR=CR-PX(7,I) PX(7,I)=CR BR=AR-PX(8,I) PX(8,I)=AR CR=BR-PX(9,I) PX(9,I)=BR AR=CR-PX(10,I) PX(10,I)=CR BR=AR-PX(11,I) PX(11,I)=AR CR=BR-PX(12,I) PX(12,I)=BR PX(14,I)=CR-PX(13,I) PX(13,I)=CR 10 CONTINUE
11	総和	$X(1)=Y(1)$ DO 11 K=2,1000 11 $X(K)=X(K-1)+Y(K)$
12	差分	DO 12 K=1,999 11 $X(K)=X(K+1)-Y(K)$
13	二次元粒子推進	DO 13 IP=1,128 $I1=P(1,IP)$

		$J1=P(2,IP)$ $P(3,IP)=P(3,IP)+B(I1+J1)$ $P(4,IP)=P(4,IP)+C(I1+J1)$ $P(1,IP)=P(1,IP)+P(3+J1)$ $P(2,IP)=P(2,IP)+P(4+J1)$ $I2=P(1,IP)$ $J2=P(2,IP)$ $P(1,IP)=P(1,IP)+Y(I2+32)$ $P(2,IP)=P(2,IP)+Z(J2+32)$ $I2=I2+E(I2+32)$ $J2=J2+F(J2+32)$ $H(I2,J2)=H(I2,J2)+1.0$ 13 CONTINUE
14	一次元粒子推進	DO 14 K=1,150 IX=GRD(K) XI=IX $VX(K)=VX(K)+EX(IX)+(XX(K)-XI)*DEX(IK)$ $XX(K)=XX(K)+VX(K)+FLX$ IR=XX(K) RI=IR $RX1=XX(K)-RI$ $IR=IR-(IR/64)*64$ $XX(K)=RI+RX1$ $RH(IR)=RH(IR)+1.0-RX1$ $RH(IR+1)=RH(IR+1)+RX1$ 14 CONTINUE

図 2.6 リバモア・14 カーネル ソースリスト

## 2.5 ベクトル・プロセッサの発展と本研究の位置付け

ベクトル・プロセッサの最大性能の動向と、1970 年から 5 年ごとに区切った発展の概要を、それぞれ図 2.7 と表 2.4 に示す<sup>2-6)</sup>。この表には、各世代ごとのベクトル演算の性能向上率とベクトル化率の概略値も記されている。

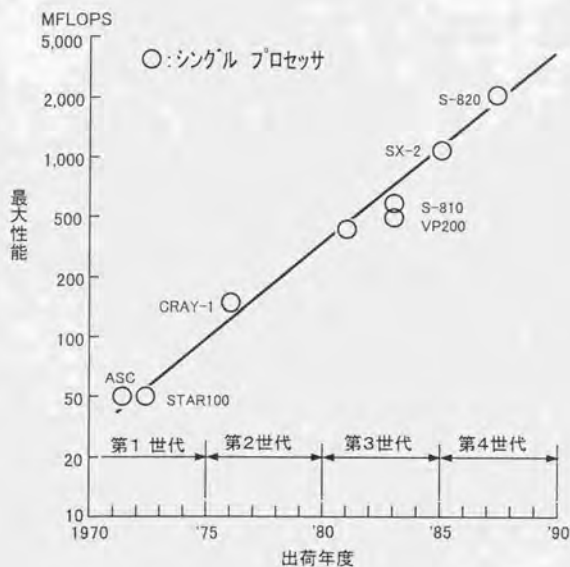


図 2.7 ベクトル・プロセッサの最大性能の動向<sup>2-6)</sup>

表2.4 ベクトル・プロセッサの発展の概要<sup>(24)</sup>

項 目		第1世代	第2世代	第3世代
年 代		～1975	1976-1980	1981-1985
最 大 性 能		～100MFLOPS	100～ 500MFLOPS	500～ 1,000MFLOPS
最大記憶 容量	主記憶	～4M バイト	～32M バイト	～256M バイト
	拡張記憶	—	—	～3G バイト
並列実行 レベル	演 算	パイプライン演算	パイプライン演算	2要素演算の並列 パイプライン演算
	命 令	—	限定のフェーシング 2 命令並列	任意のフェーシング 2～4 命令並列
	プロセッサ	—	—	2CPU 構成 マルチプロセッサ
ベクトル演算性能向上率		～20	20～50	50～100
ベクトル 演算種類	四則演算	○	○	○
	マクロ演算	○	○	○
	条件付演算	×	○	○
	間接指標	×	×	○
ベクトル化率の概略値		～50%	～60%	～80%

以上に述べてきたこと、およびこれらの図表から、ベクトル・プロセッサの高速処理のための基本要素は、次の2点であることがわかる。

- ・ベクトル演算処理対象の拡大
- ・ベクトル演算の高速化

これらに関して、従来技術では以下に述べるような課題があった。

#### (1)ベクトル演算処理対象の拡大

第1世代のベクトル・プロセッサでは、ベクトル演算処理の対象は主として四則演算およびマクロ演算であった。実際にはこれでは典型的な科学技術計算プログラムの実行ステップ数の約50%しかベクトル演算処理の対象とはならなかった<sup>27)</sup>。これでは、ベクトル演算処理の対象部分が非常に高速に処理されたとしても、全体の性能は高々2倍弱である。第2世代および第3世代はベクトル演算処理対象を拡大させるために、プログラムの分析



と、ベクトル命令セットの拡充に努力がはらわれた。すなわち、第2世代では条件付演算などが、ベクトル演算処理が可能になり、第3世代では間接指標ベクトルなどが、ベクトル演算処理が可能になった。これらにより、第1世代ではベクトル演算処理の対象が約50%であったものが、第3世代では約80%にまで向上した。

## (2) ベクトル演算の高速化

ベクトル演算による大幅な性能向上は、各種のレベルで処理を並列化することにより実現されている。この処理の並列実行方式は表2.4によれば次のようにまとめられる。

- ・演算の並列実行
- ・ベクトル命令の並列実行
- ・マルチプロセッサによる並列処理

演算の並列実行は、基本的にはパイプライン演算方式を採用しているが、第3世代から複数のベクトル要素を同時に演算する要素並列パイプライン演算方式が開発され、2要素程度の並列演算が可能になってきている。

ベクトル命令の並列実行は、第2世代になって主記憶と演算器の間にベクトル・レジスタが導入された。この目的は、演算途中の結果を毎回主記憶に戻すことを不要にしたり、主記憶から読み出したデータを再利用したりするためであった。さらに、ベクトル・レジスタを利用することによって、前の命令の演算がすべて完了する前に次の命令が開始できる、すなわちベクトル命令の並列実行ができるチェイニング方式が開発された。

しかし、この世代のチェイニングは前後の命令間の時間関係に制約があり、並列実行ができる場合が限定されていた。これに対して、第3世代になると、前の命令の実行状態に無関係に後続命令のチェイニングが実行できる動的チェイニング技術が確立し、命令の並列実行度が大幅に向上した。

マルチプロセッサによる並列処理は、第3世代から一部のベクトル・プロセッサに導入された。この方式はベクトル演算の性能を向上させるとともに、ベクトル演算処理が不可能な部分についても並列処理ができる可能性を持っている。しかしながら、ベクトル演算処理の性能に関しては、ハードウェアで制御を行う要素並列パイプライン演算方式の方がオーバーヘッドが少なく、高速処理が得られる。また、演算や命令レベルの並列実行がハードウェアやコンパイラによってユーザに負担をかけることなく実現されているのに対し、プロセッサレベルの並列処理は、ユーザがなんらかの指示をする場合が多く、使い易さ及び性能の点から問題がある。

このため、本研究の目的は上記の課題を解決する技術を開発することにある。本論文では特に、シングル・プロセッサのベクトル演算処理の高速化に関し、演算の並列実行を高速に処理する「複数のベクトル要素を同時に演算する要素並列パイプライン演算方式」および「要素間に相互依存性のある処理の高速化方式」を中心に述べる。これらはいずれも第4世代のマシンで初めて開発され、あるいは本格的に実用化された技術であり、1987年12月に出荷した HITAC S-820 の性能の大きな特長となっている。

## 2.6 ベクトル・プロセッサの発展とコンパイラ技術との関連

ここではベクトル・プロセッサの発展とコンパイラ技術との関連について述べる。

図2.7と表2.4でベクトル・プロセッサの発展を1971年から5年単位で第1世代～第3世代に分けた。この世代間におけるコンパイラ技術との関連を表2.5に示す。

表2.5 ベクトル・プロセッサの発展とコンパイラ技術との関連

演算要素種類	第1世代 (71～75年)	第2世代 (76～80年)	第3世代 (81～85年)
1 四則演算	ソースレベルでのライブラリコール	自動ベクトル化	自動ベクトル化
2 条件付き演算	ベクトル命令なし	ソースレベルでのライブラリコール	自動ベクトル化
3 内積・総和演算	ソースレベルでのライブラリコール	ソースレベルでのライブラリコール	自動ベクトル化
4 1次巡回演算	ベクトル命令なし	ベクトル命令なし	自動ベクトル化
5 積和演算など	ベクトル命令なし	ベクトル命令なし	自動ベクトル化
6 間接指標演算	ベクトル命令なし	ベクトル命令なし	自動ベクトル化

第1世代は、アーキテクチャとしてもベクトル命令が四則演算や内積・総和演算に限られていた。コンパイラ技術もまだ自動ベクトル化ができておらず、ソースプログラムの中でライブラリ・コールによってベクトル演算を処理していた。

第2世代になって自動ベクトル化が実現し、ベクトル・プロセッサが急激に普及していく状況になったが、条件付き演算や内積・総和演算はまだ自動ベクトル化ができず、ライブラリ・コールによるベクトル演算処理であった。

第3世代になると、多くの科学技術計算分野へのベクトル・プロセッサの普及に伴い、1次巡回演算、積和演算、間接指標演算などのベクトル命令がサポートされ、コンパイラもこれらの処理を自動ベクトル化可能にした。

第4世代では、第3世代でアーキテクチャの点からはほぼ完成の域に達したベクトル命令を、さらに高速化する技術の開発が課題である。この論文は主としてアーキテクチャおよびハードウェアの論理方式に関して述べるが、関連するコンパイラ技術に関しても主としてアーキテクチャと性能向上の点から記述する。

ベクトル・プロセッサの発展と普及は、アーキテクチャ・ハードウェア・ソフトウェアが表裏一体となっている。主としてアーキテクチャ・ハードウェアが先行し、コンパイラ技術による広範囲にわたる自動ベクトル化技術とその高速化技術のサポートにより、ベクトル・プロセッサの発展と普及がある。またこの間、多くのユーザからの意見が寄せられ、使い勝手に関する改善（例えば高速化プログラムのためのチューニング・ツールのサポートなど）も数多く行われている。

### 第3章ベクトル処理方式の分析と問題点

#### 3.1 ベクトル処理方式の概念

まず現在のスーパーコンピュータの高速化方式として広く採用されているベクトル処理方式の概念と高速化できる理由について説明する。

図 3.1 はベクトル演算の概念と高速化の理由について示したものである。一般に科学技術計算では

$$A_i = B_i \times C_i + D_i \quad (i=1 \sim N)$$

のような演算、すなわち  $i$  を一定値ずつ増加させながら繰り返す繰返し演算が多用される。

このような繰返し演算を従来の汎用計算機では図 3.1 の上段のように処理していた。すなわち、1つの  $i$  の処理は  $B_i$  の主記憶からの読み出し、 $B_i$  と  $C_i$  との乗算、この積と  $D_i$  との加算、結果の  $A_i$  への格納、 $i$  の更新、 $i$  が  $N$  に到達したかどうかのチェックと処理の繰返しのための分岐からなる。この処理に13サイクルを要する。(M-280Hでの例。なお汎用計算機では命令がパイプライン処理で実行されるが、ここに示した図は各命令の演算ステージのみを取り出した図である)。

一方ベクトル演算方式では次のように行う。まず  $A_i$  ( $i=1 \sim N$ ) を一つにまとめてベクトル  $A$  と考える。同様にベクトル  $B$ 、 $C$ 、 $D$  を考えると、先の演算は

$$A = B \times C + D$$

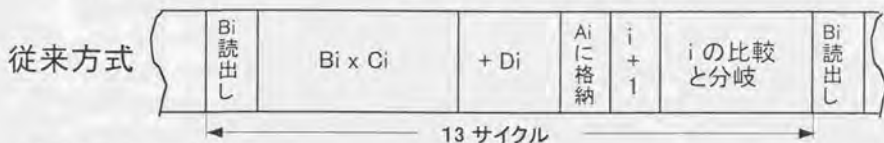
とすることができる。処理にあたってはまずベクトル  $B$  の要素である  $B_1, B_2, B_3, \dots$  を次々と主記憶からの読み出す。同様にベクトル  $C$  の要素も読み出し、両ベクトルの各要素を次々と乗算していく。以降の演算も同様である。ある  $i$  に着目してみると、従来方式のようにデータの読み出し、乗算、加算、格納などが行われるが、次の  $i$  は1サイクル後に、その後の  $i$  はさらに1サイクル後に、図 3.1 に示すように実行することが可能である。

これが実現できる理由は次の点にある。

- (1) 主記憶上のベクトルデータ配列の規則性：ベクトル要素が主記憶上に規則正しく格納されている。
- (2) 演算の画一性：すべてのベクトル要素に同一演算を施す。
- (3) ベクトル要素間の独立性：あるベクトル要素  $i$  と他のベクトル要素  $j$  の間は独立に演算することができる。



繰返し演算方式:  $A_i = B_i \times C_i + D_i \quad (i = 1 \sim N)$



[スカラー演算方式]

ベクトル演算方式:  $A = B \times C + D$



[パイプライン演算方式]      . . . . .

図3.1 ベクトル演算の概念と高速化の理由



ベクトル演算では基本的には1つのiの処理は1サイクルで完了させることができる。この場合は従来方式と比較して13倍高速に処理することになる。なお、iの更新、iがNに到達したかのチェックは演算と並列にハードウェアが行うので、実行時間としてこれが現れることはない。このように次々と処理する方式をベクトル演算方式、またはパイプライン演算方式と呼ぶ。また従来の処理方式のことをベクトルという言葉と対比してスカラ演算方式と呼ぶことがある。

### 3.2 S-810におけるベクトル演算処理の実現方式

本格的なベクトル演算方式を採用した HITAC S-810 (1983 年 10 月出荷)におけるベクトル演算処理の実現方式を図 3.2 に示す。主記憶、ベクトル処理ユニット、入出力処理装置、拡張記憶からなる。ベクトル命令を実行するベクトル処理ユニットは、6 個のパイプライン演算器を中心にアドレス・レジスタ、ベクトル・ロード、ベクトル・ストア、スカラ・レジスタ、ベクトル・レジスタ、マスク・レジスタなどからなる。

またベクトル処理ができない部分は、従来の命令で実行することになるが、このためにスカラ処理ユニットを設けた。

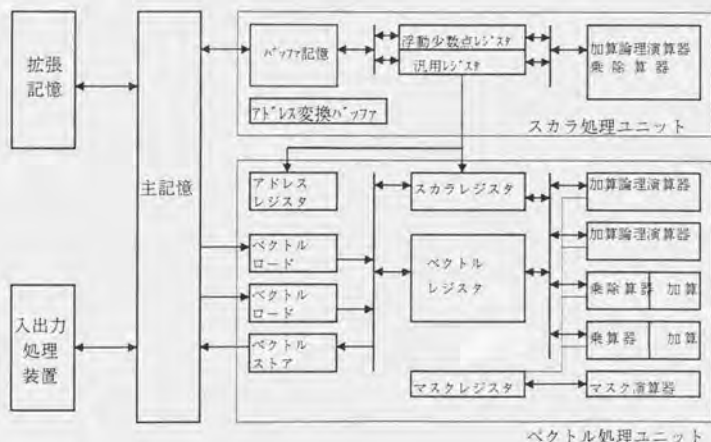


図 3.2 HITAC S-810 の論理構造概略図

高速のベクトル演算処理を実現するために、特にベクトル処理ユニットを設けた。そこではパイプライン演算器を複数、並列に動作させる方式を採用した。S-810 モデル 20 では、加算論理演算器が 4 個、直列に接続する乗除算器と加算器が 2 個、おなじく直列に接続する乗算器と加算器が 2 個、合計 12 個のパイプライン演算器を並列に動作できるようにすることにより最大処理能力を高めている。

この中で、直列に接続する乗算器と加算器を設けた理由は、数値計算の解法では、大型の科学技術計算に反復解法（準陰解法や陽解法）が使用されることが多い。この場合、中心となる演算は多項の積和計算である。この計算に含まれる乗算や加算の演算子ごとにパイプライン演算器を割り付けて、複数の演算器を連結させた形で並列に処理することにより、高速にベクトル処理ができるためである。

### 3.3 S-810 の課題と S-820 の開発技術

S-810 は出荷当時は最高性能のベクトルプロセッサであったが、いくつかの課題が残された。S-810 の後継機種である S-820 ではこの改善をはかるとともに、さらに高性能を目指して技術開発を進めた。表 3.1 は S-810 の課題、S-820 の方針と開発技術について要約したものである。

表 3.1 S-810 の課題と S-820 の開発技術

	S-810 の課題	S-820 の方針	S-820 の開発技術
1	項数の少ない演算での性能向上	項数に依存しない安定した演算性能の実現	要素並列演算パイプライン方式
2	短ベクトル演算性能の向上	ベクトル処理起動オーバーヘッドの短縮	ベクトル処理分割起動方式
3	TSS 環境でのベクトル処理の実現	アドレス変換と拡張記憶利用による TSS のサポート	高速アドレス変換方式
4	スカラ性能の向上	高速プロセッサ M-680 の利用	

課題の第 1 は  $A=B+C$  のような項数の少ない演算での性能向上である。次節でも述べるように、S-810 では演算式中の乗算、加算のような各演算子に 1 つの演算器を割り付ける方式を採用した。このため項数の多い演算のときには多数の演算器が稼動し高い性能を発揮することができたが、項数の少ない場合には演算器が十分稼動しないという問題があった。

これに対し S-820 では項数に依存しない安定した演算性能を実現することとし、要素並列型演算パイプライン方式と呼ばれる技術を開発した。

第2の課題は処理ベクトル要素数が少ない、いわゆる短ベクトル時の演算性能の向上である。S-810 ではベクトル演算をベクトル準備処理とベクトル処理に分け、ベクトル処理をベクトル処理ユニットで、ベクトル準備処理（ベクトルアドレスやベクトル要素数のベクトル処理ユニットへのセット等）をスカラ処理ユニットで行うこととし、ベクトル処理の開始にあたっては、そのベクトル処理に必要な準備処理をすべて完了してからベクトル処理を開始するようにした（開始後スカラ処理ユニットは次のベクトル処理の準備処理ができるようにした）。

しかし、当初想定したよりもベクトル準備処理をベクトル処理とオーバーラップさせることができず、準備処理がオーバーヘッドとなって処理要素数が少ない場合のベクトル処理性能が十分でないという問題があった。これに対しは S-820 ではベクトル処理起動のオーバーヘッドを短縮させるために、ベクトル処理と準備処理をオーバーラップさせるベクトル処理分割起動方式技術を開発した。

課題の3番目は TSS 環境でベクトル処理ができないという点である。S-820 はベクトル処理での高速アドレス変換（アドレス・リロケーション）を実現し、オペレーティング・システムでジョブ全体を拡張記憶と主記憶の間でロール・イン/ロール・アウトする機能を開発することによって、TSS 環境でのベクトル処理を実現することとした。

課題の4番目はスカラ処理性能の向上である。S-820 ではスカラ処理ユニットとして当時の最新の M-680H を母体とし、S-810 のスカラ処理ユニットに対し2倍以上の高速化を図った。

本論文では特に S-820 のベクトル演算性能向上に大きく貢献した要素並列型演算パイプライン方式および要素間に相互依存性のある処理の高速化方式とベクトル処理の分割起動方式についてその詳細を述べる。

## 第4章 要素並列型演算パイプライン方式

### 4.1 S-820 におけるベクトル演算処理方式

S-810 を製品化することにより、国産のスーパーコンピュータは多くのユーザに使用された。われわれの S-810 の分析と共に、S-810 を使用したユーザからの要望事項をフィードバックして S-820 は製品化された。図 4.1 に HITAC S-820 の論理構造の概略図を示す。

ベクトルプロセッサ (S-810 ではベクトル処理ユニット)、スカラプロセッサ (S-810 ではスカラ処理ユニット)、主記憶、拡張記憶、入出力装置といった大きな構成単位は、S-810 と同じであるが、S-810 と比較して下記の論理方式上の特長を持つ。

#### (1) スカラプロセッサ

内部のスカラ演算器 (加算論理演算器と乗除算器) にパイプライン処理を開発した。これにより、スカラプロセッサも 1 マシンサイクル ピッチで演算結果が得られるようにした。またベクトルプロセッサに対する起動をきめこまかく行うことにより、ベクトル処理起動のオーバヘッドを短縮させ、ベクトル処理と準備処理をオーバーラップさせるベクトル処理分割起動方式技術を開発した。

#### (2) ベクトルプロセッサ

ベクトル演算における、演算器の使用効率を高めるために要素並列型演算パイプライン方式を開発した。このため、図 4.1 に示すように、加算論理演算器、乗算器・加算器はそれぞれ要素数に対応して並列に使用される。図において、太線の部分は要素並列型パイプライン処理の部分である。演算器に対応して、ベクトル・レジスタおよび主記憶とのデータのやり取りを行うベクトル・ロード、ベクトル・ロード/ストアも要素数に対応して並列化した。

これにより、要素並列処理が可能な演算は高速化を図ることが可能になるが、ベクトル演算の中には、要素並列処理が不可能な演算もある。例えば、内積、総和型の演算や巡回型の演算などである。前者の内積、総和型の演算は部分積や部分和を演算する処理は、要素並列が可能な演算であるが、最後にまとめる部分は要素並列ができない。この部分の高速化を図るために、乗算器・加算器に接続する後処理演算器を新しく設けた。



要素並列型演算パイプライン方式の考え方は、複数の演算器を並列に動作させることから主記憶共有型のマルチプロセッサでは、ソフトウェア的に処理することも CRAY-XMP などで考えられたようではあるが、実現には至っていない。

国産メーカーではシングルプロセッサの性能を向上させるために、要素並列型演算パイプライン的な考え方を取り入れてきた。S-810 では部分的に限定された命令で2要素並列の処理を行っているが、これは要素並列型演算パイプラインが不可能な命令の処理に十分なハードウェアをかけられなかったためである。後処理演算器の導入も含めて要素並列型演算パイプラインの高速方式を実現したのはS-820が初めてである（S-820の要素並列型演算パイプライン制御方式が特許として成立している）。

### (3) 拡張記憶

ベクトルプロセッサの演算速度は十分速い。この速度に対応して、大量のデータの入出力速度も上げなければならない。このため、主記憶とディスク装置の間に、半導体メモリを搭載した拡張記憶を加えた。この拡張記憶はS-810にもあり、主記憶と拡張記憶との間のデータ転送はスカラプロセッサから発行される同期型転送命令により、制御される。これに対し、S-820では同期型転送命令に加えて非同期型転送命令を追加した。非同期型転送命令は、スカラプロセッサがその命令を発行すれば、あとはプロセッサの介在なしでデータを転送できる。オペレーティング・システムがベクトル・ジョブのスワッピング制御を行う場合などに使用する。

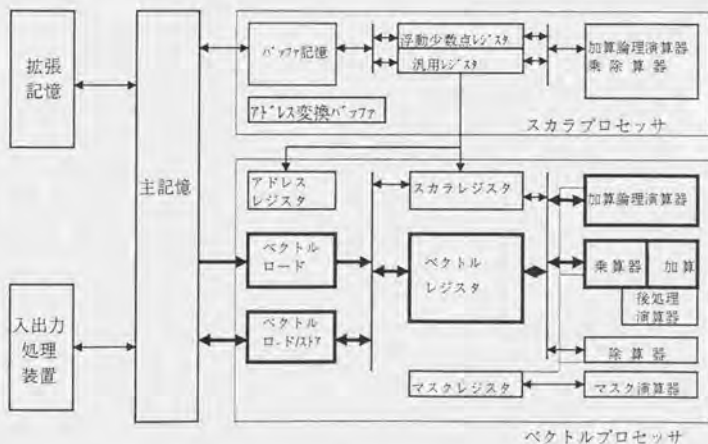


図 4.1 HITAC S-820 の論理構造概略図  
(太線は要素並列型パイプライン処理の部分)



## 4.2 要素並列型演算パイプライン方式

前述のように S-820 では演算器の利用率を向上させるために要素並列型演算パイプライン方式を開発した。本方式の詳細を述べる前に、S-810 の演算パイプライン方式について評価してみる。

図 4.2 は S-810 の演算パイプライン方式を示したものである。ベクトルは主記憶に格納されており、ベクトル処理ユニットから読み出されて演算され、結果が再び主記憶に格納される。S-810 のベクトル処理ユニットにはベクトルを主記憶から読み出すロード回路や主記憶に格納するストア回路 (S-810 ではストア回路はロード回路と兼用になっており、一時にはロードかストアのいずれかを処理することができる)、ベクトル・データを一時蓄えるベクトル・レジスタ、加算器、乗算器、およびこれらを接続する書き込みスイッチ、読み出しスイッチから構成される。

ベクトル処理時にはベクトル・データは主記憶からロード回路によって読み出されてベクトル・レジスタに蓄えられる。これが読み出されて演算され、結果が再びベクトル・レジスタに戻される。演算が終了すると演算結果がストア回路によって主記憶に格納される。

S-810 には、6つのロード回路、2つのロード/ストア兼用回路、4つの加算器、4つの乗算器がある。それぞれを図 2.3 中に示すように番号付けすると、先に示した  $A=B \times C + D$  の演算を行う場合には、演算器 (ロード/ストア回路も広義意味での演算器と呼ぶこととする) の割り当ては次のようになる。

ベクトル B, C, D の読み出し	ロード回路①、②、③
乗 算	乗算器⑤
加 算	加算器⑥
ベクトル A 領域への格納	ストア回路⑦

すなわち、S-810 では1つのベクトルの参照 (読み出しまたは書き込み)、および1つの演算子に対し1つの演算器を割り当てる。このようにした背景には、一般に繰返し演算 (実際には Fortran プログラムの DO ループ) には実際には一つの式ではなく複数の式が存在する場合が大部分であること (従って複数の式のベクトル、演算子を割り付けることによって演算器の利用率を上げることが可能なこと)、仮に少ない場合にはコンパイラによって複数の演算式に展開したり、図中の番号の奇数番と偶数番を対にし、ベクトル要素の  $i$  番と  $i+1$  番を同時に実行できるモード (ペア・モード) を設けることによって利用率の向上が可能であると考えたためであった。

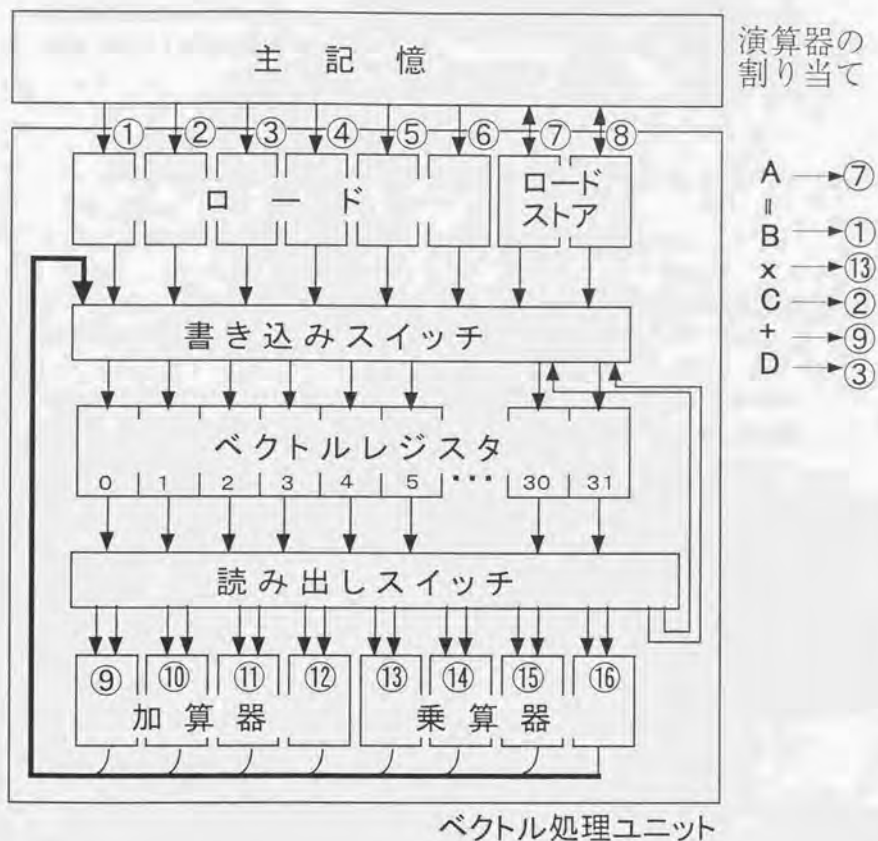


図4.2 S-810の演算パイプライン方式

実際のプログラムでの平均利用率は(詳細は後述するが)、リバモアーループ(米国 Lawrence Livermore 研究所で作成されたスーパーコンピュータの評価用ベンチマーク。実効性能に標準的に利用されるもので14ループから構成される)中のベクトル化された8ループでの場合、次のようであった。

ロード/ストア回路	0.36
加 算 器	0.38
乗 算 器	0.38

ただしここではコンパイラでの工夫やベア・モードでの実行は含めていない。

これに対して S-820 では要素並列型演算パイプライン方式を開発することによって演算器の利用率を向上させることとした。この方式の詳細を図 4.3 に示す。

S-820 では加算器、乗算器、ロード/ストア回路とも S-810 の2倍、加・乗算器をそれぞれ8個、ロード専用回路を8個、ロード/ストア兼用回路を8個設けることによって高性能を実現している。このとき、それぞれの演算器は図 4.3 に示すように  $i, i+1, i+2, \dots, i+7$  と連続する8要素を並列に、同時に演算する(このため要素並列型演算パイプラインと呼ぶことにした)。主記憶とベクトル・レジスタ間のベクトル・データ転送のためにも8要素並列のロード回路と8要素並列のロード/ストア回路をそれぞれ1つずつ設けている。

加算器、乗算器、ロード/ストア回路とも S-810 の2倍にした理由は、「シングル・プロセッサで世界最高性能を実現」することにあつた。S-810 では1500ゲート・ゲート遅延0.5nsのバイポーラ・ゲートアレイを使用していたが、S820では5000ゲート・ゲート遅延0.25nsのバイポーラ・ゲートアレイを使用することができたため、これらのハードウェアを2倍にすることは可能であった。

また8要素を並列に演算を行うと、ベクトル長との関係が問題になる。たとえばベクトル長が64であれば、8要素並列演算では実効的なベクトル長は8となる。16要素並列演算では実効的なベクトル長は4となってしまう。プログラムを解析するとベクトル長は数十から数千まで分布するが、大規模計算では64以上と考えてよいであろう。ベクトル演算時間はサイクル数で表わすと

- $\alpha + L/8$  : 8要素並列演算の場合  $\alpha$  はベクトル前処理時間、 $L$  はベクトル長  
 $\alpha + L/16$  : 16要素並列演算の場合

なる。ベクトル長が短いところでは、この値が特に敏感である。ベクトル前処理時間  $\alpha$  を小さくすることは、非常に重要である。S-820 ではアーキテクチャとハードウェアの改善により、 $\alpha$  を 10 サイクル程度にすることが可能であった。従って、 $L=64$  では

$$8 \text{ 要素並列演算の場合: } \alpha + L/8 = 10 + 8 = 18$$

$$16 \text{ 要素並列演算の場合: } \alpha + L/16 = 10 + 4 = 14$$

となり、16 要素並列演算の効果があまりない。このためハードウェア量と性能の関係から加算器、乗算器、ロード/ストア回路とも S-810 の 2 倍にした。

S-820 ではこの 8 要素並列の処理を 8 ns ですなわち等価的には 1 ns に 1 演算ずつできるようになっている (1 演算/ns は 1 GFLOPS: Giga Floating-point Operations Per Second となり、S-820 では乗算と加算が同時に実行できるので最大性能が 2 GFLOPS となる)。

演算  $A=B \times C + D$  での演算器の割り当ては図 4.3 の右側に示すようになる。すなわち、

- ・ロード回路①、ロード/ストア回路②でベクトル B、C を読み出して乗算器④で乗算し、結果をベクトル・レジスタに格納する。この詳細を図 4.3 に示す。
- ・次にロード回路①でベクトル D を読み出し、ベクトル・レジスタ中にある積と加算器③で加算し、ロード/ストア回路②で再び主記憶に格納する。

この時の演算器の利用率はロード回路①、ロード/ストア回路②は 100%、加算器、乗算器はそれぞれ 50% となる。

また S-820 におけるリバモアループでの平均利用率は、ベクトル化された 8 ループでの場合、次のようであった。

ロード/ストア回路	0.82
加 算 器	0.75
乗 算 器	0.73



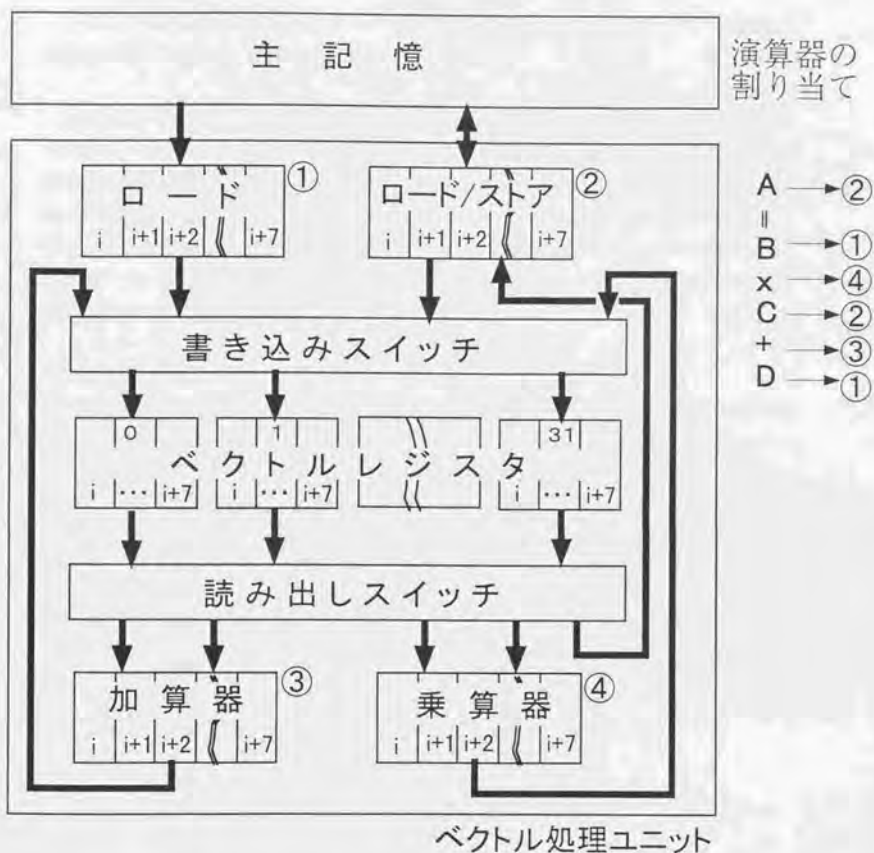


図4.3 要素並列型演算パイプライン方式



#### 4.3 要素並列型演算におけるベクトル・レジスタ構成

このような要素並列型演算パイプライン方式の実現に当たっては、ベクトル・レジスタの構成に関して次に述べるような工夫が必要であった。

32本の各ベクトル・レジスタを8要素並列で実現しようとする、物理的には256本のベクトル・レジスタが必要となる。各レジスタはさらに読み出しと書き込みを同時に実行させる必要があるため2バンク制御になっており、従って、バンク単位で数えると512ものバンクとなる。仮に1つのバンクを256ワード×nビットのメモリ素子で構成したとすると、1本のベクトル・レジスタに含まれるベクトル要素数は $256 \times 2 \times 8 = 4096$ となり、通常扱うベクトル長(100～数100)からみても必要以上に長くなってしまふ(不要のメモリを多数持つことになる)。

この解決として次のようにした。

- (a) ベクトル・レジスタは4ns単位で読み書きが可能なベクトル・レジスタ専用の論理混載型高速RAM(一つのRAMは128ワード×9ビット構成)を開発することとし、これによって8要素並列ではなく、4要素並列ですませることとした。
- (b) 1つのメモリ素子内に2本分のベクトル・レジスタを入れることにした。この場合同時に同一チップ内の2本のベクトル・レジスタの読み出し・書き込みは不可能であるが、ベクトル・レジスタ番号の割付けを工夫することによって、従来機種(S-810)でのベクトル・レジスタの番号付けと矛盾が生じないようにした。

この結果、S-820のベクトル・レジスタのベクトル要素数は512と、通常使用する範囲に収めることができた。図4.4に要素並列型演算におけるベクトル・レジスタの構成および演算器との対応を示す。

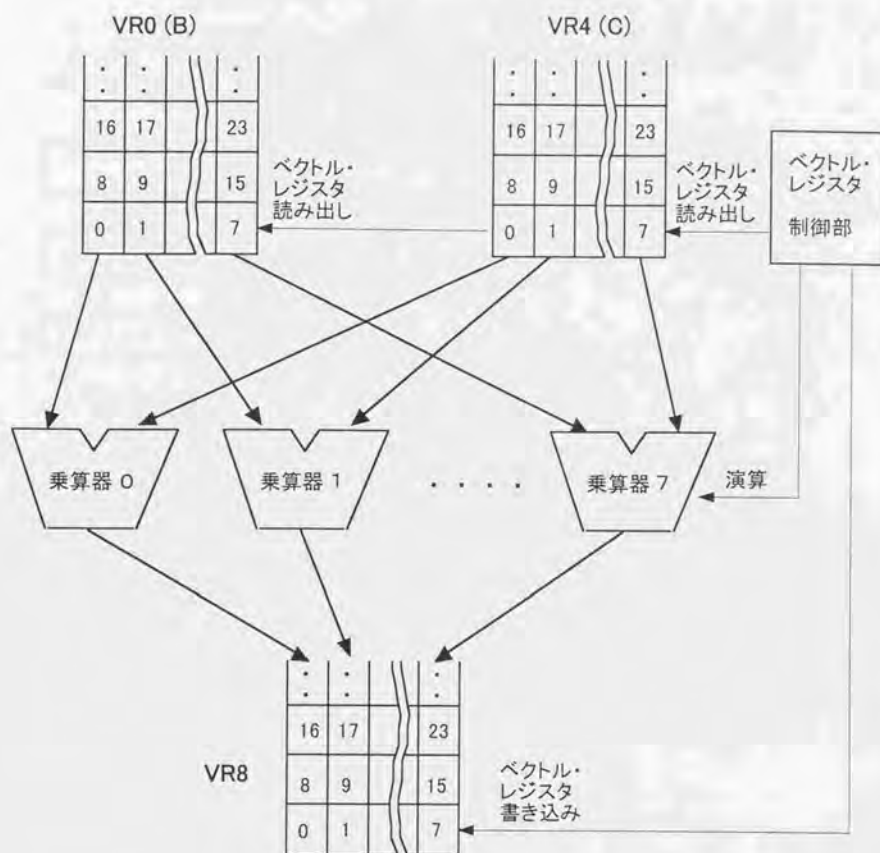


図4.4 要素並列処理の例  
(  $VR8 = VR0(B) \times VR4(C)$  ) の例)

#### 4.4 ベクトル演算制御の実現方式例

ここではベクトル演算制御の具体的な実現方式例として S-820 の例について述べる。図 4.5 に S-820 のベクトル演算制御の構造を示す。

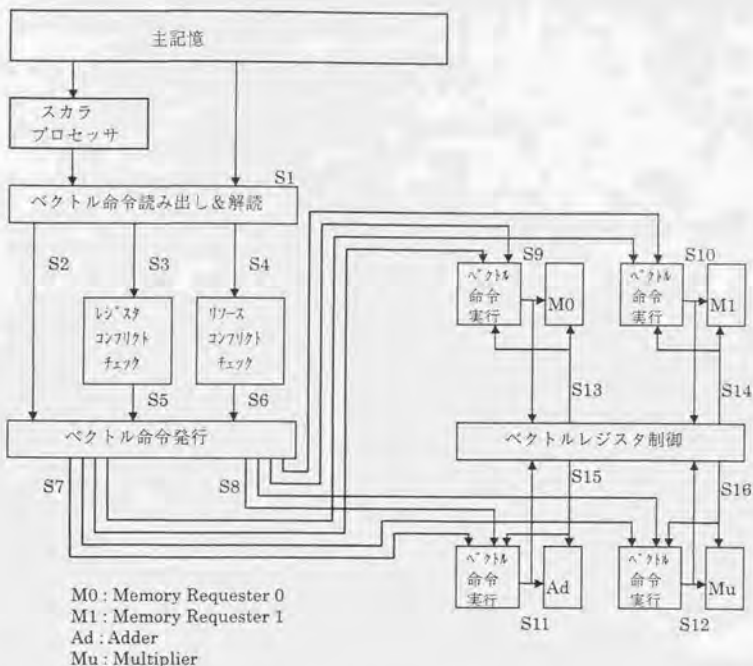


図 4.5 S-820 のベクトル演算制御の構造

主記憶にベクトル命令とそのデータが格納されており、S1 のベクトル命令読み出し＆解読部が主記憶よりベクトル命令の読み出しを行い、解読を行う。レジスタ・コンフリクト・チェック部は該ベクトル命令で使用するベクトル・レジスタと既発行のベクトル命令で使用するベクトル・レジスタとの間でのコンフリクトのチェックを行う。

また、該ベクトル命令で使用するベクトル命令実行部でのリソース・コンフリクト・チェックを行う。これはベクトル命令実行部は大きく 4 ケの部分からなり、それら M0 : Memory Requester 0, M1 : Memory Requester 1, Ad : Adder, Mu : Multiplier をリソースと呼ぶ。既発行のベクトル命令で使用しているリソースとのコンフリクトのチェックを行う。

このようにして解読されたベクトル命令は、レジスタおよびリソースのコンフリクトが無いことを確認してベクトル命令実行部へ送られる（これをベクトル命令発行(issue)と呼ぶ）。ベクトル命令実行部へ送られる情報は、ベクトル命令の起動信号(S7)、命令コードおよびベクトル・レジスタ番号など(S8)である。

レジスタおよびリソースのコンフリクトが発生した場合は、ベクトル命令は解読部でコンフリクトが解除されるまで待たされる。

ベクトル命令実行部での各リソースの役割は、

- ・ Memory Requester : 主記憶に対するリクエスト処理
- ・ Adder : 加算器処理
- ・ Multiplier : 乗算器処理

である。それぞれのリソースは、8つのベクトル要素を並列に処理する。またこれらの4つのリソースは、一般にはそれぞれ異なるベクトル命令を処理する。

ベクトル命令実行部は対応するリソースにS9, S10, S11, S12のパスを経由して制御情報を送り、各リソースの処理の起動を行う。起動された各リソースはベクトルレジスタ制御部と通信を行い、ベクトル・レジスタとの読出し/書込み制御を行いながら、各リソースの処理を進めて行く。

図 4.6 に  $A=B \times C + D$  のタイムチャート概略を示す。

この例に示されているように、最初の3命令（ロードB、ロードC、乗算 $B \times C$ ）は命令解読、コンフリクトチェック、命令発行と1サイクル毎に進む。乗算 $B \times C$ の演算実行はロードB、ロードCのデータが主記憶よりベクトル・レジスタに格納した後に行われる。ロードDはMemory Requester 0が使用中であるため、リソース・コンフリクトが発生し、リソースが空くのを待って命令発行が発行される。

加算(+D)は命令解読、コンフリクトチェック、命令発行と1サイクル毎に進むが演算実行はロードDのデータが主記憶よりベクトル・レジスタに格納した後に行われる。

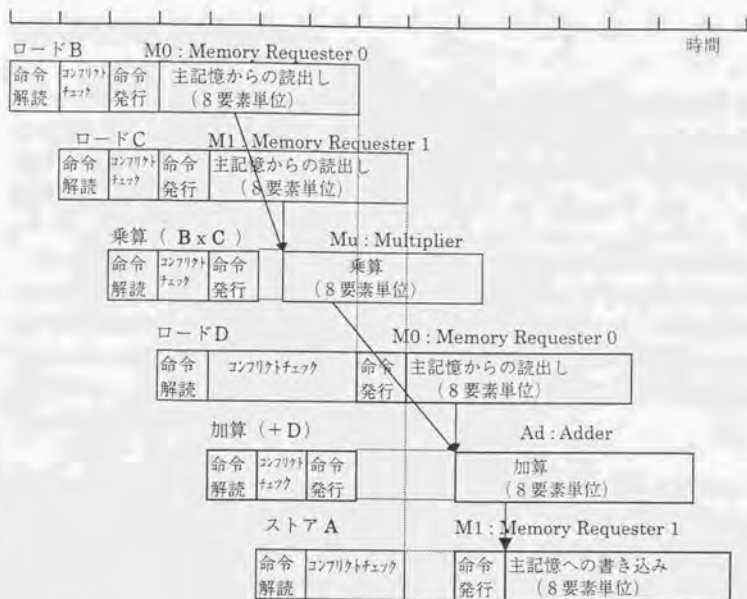


図 4.6 ベクトル命令制御タイムチャート概略  
(  $A = B \times C + D$  の例 )



#### 4.5 コンパイラによる高速処理方式

アーキテクチャの点からはほぼ完成の域に達したベクトル命令を、さらに高速化する技術の開発が課題である。この論文では主としてアーキテクチャおよびハードウェアの論理方式に関して述べているが、ここでは性能向上の点から関連するコンパイラ技術に関して記述する。

ベクトル化された DO ループに関しては、基本的には高速化方式は、ハードウェアによるものと、コンパイラによるものとに分類できる。従来は DO ループをコンパイラがベクトル命令に展開し、ハードウェアがそのベクトル命令を高速に処理するという役割であった。しかしコンパイラ技術の進歩により、ハードウェアの特性（例えばベクトル長が長く、演算数が多く、主記憶アクセスが少ない方が高速化されるなどの特性）を最大限に活かすプログラム変換技術などが可能になった。

筆者は多くの性能向上対策項目に関して、ハードウェアおよびソフトウェアの両面から検討し、実現する場合はそのどちらで行うかを決定した。ハードウェアで行った項目については、主設計者として実現し、この論文の主たる部分として述べている。コンパイラによる次の高速化項目については、筆者は性能向上対策項目として検討に参加したが、実現はコンパイラ担当者が行った。

##### 4.5.1 ベクトル命令列が短い場合の高速化

主記憶上のベクトル要素に対する参照リクエストが送出されてから、その要素がベクトル・レジスタに書き込まれるまでの時間をベクトル・ロード命令のトラベルタイムと呼ぶ。ベクトル命令列が短い場合、ソフトウェアによるループ制御（ベクトル長を例えば 256 単位に区切って処理すること）を行うと、このトラベルタイムが性能に見えてくることがある。すなわち次のループでは同じベクトル・レジスタに読み書きを行うので、コンフリクトにより命令の発行が抑止される。一般にベクトル命令列の最初はロード命令であることが多いので、この場合性能は非常に低下する。

例えば、 $A=B+C$  などの単純演算のみを行う DO ループでは、ベクトル命令列は短い。このような場合の性能低下を防ぐために、コンパイラ技術によりベクトル・レジスタのアサインのみが異なる全く同じベクトル命令列を 2 つ用意し、交互に起動する方法を採用した。この方法を図 4.7 に示す。



図 4.7  $A(i)=B(i)+C(i)$  のコンパイラによる高速化方式

#### 4.5.2 プログラム変換による高速化

DO ループ内のベクトル処理の性能をより向上させるために、コンパイラによるプログラム変換による高速化技術を開発した<sup>46)</sup>。これらは大別すると

- (1) DO ループ長の長大化
- (2) DO ループ内演算数の増加
- (3) DO ループ内ベクトル共通オペランドの増加
- (4) 条件文処理の効率化

になる。ここで述べるプログラム変換のすべては、一定の条件が整えばコンパイラによる自動プログラム変換がなされている。

## [1] ループ長の長大化

ベクトル処理の性能をより向上させるためには、ループ長を大きくすることが効果がある。このための方法として、多重ループの場合、「ループ交換」と「ループ重化」によってループ長を大きくすることができる。

### (a) ループ交換

この例に示すように、多重ループにおいて、最内側ループ DO 20 の外側にある DO 10 の方のループ長が大きいために、ベクトル化をより大きなループ長をもつ DO 10 で行うためにループを入れ替えることをループ交換と呼ぶ。

```
DO 10 I=1,1000
DO 20 J=1,10
20  A(I,J)=B(I,J)*C(I,J)
10  CONTINUE
    ↓
DO 20 J=1,10
DO 10 I=1,1000
10  A(I,J)=B(I,J)*C(I,J)
20  CONTINUE
```

### (b) ループ重化

この例に示すように、複数のループにまたがって参照される配列要素のアドレス増分が一定のときは、複数のループを1つのループに構成し直すことができる。この例では、制御変数 (I,K) により、配列 A(J,I,K) と配列 B(I,K) が1つの制御変数によりアクセスが可能であり、その際、配列 A は等間隔参照、配列 B は連続参照となる。

```
DIMENSION  A(10,20,30), B(20,30)
```

```
DO 10 I=1,20
DO 20 J=1,5
DO 30 K=1,29
30  B(I,K)=A(J,I,K)*S
20  CONTINUE
10  CONTINUE
    ↓
```

```

DO 20 J=1,5
DO 50 L=1,580
50  B(L,1)=A(J,L,1)*S
20  CONTINUE

```

(c) リストベクトル使用によるループ重化

リストベクトルの使用によるループの一重化は、次の例のように、もともとの多重ループのインデックスを新たな1次元インデックスの関数として表現することで実現できる。しかし、このままではリストベクトルを生成する部分のオーバーヘッドが大きく、性能が逆に悪化する。ところが、このプログラム部が何回も呼ばれるときや、インデックスの範囲が共通のループがあるときは、リストベクトルを1回だけ生成すればよいので、一般に性能が向上する。

```

DO 10 J=1,M
DO 20 I=1,N
20  A(I,J)=B(I,J)+1.0
10  CONTINUE
      ↓
      K=0
DO 30 J=1,M           ;リスト生成部
DO 20 I=1,N
K=K+1
LI(K)=I
20  LJ(K)=J
30  CONTINUE

DO 10 L=1,K
I=LI(L)
J=LJ(L)
10  A(I,J)=B(I,J)+1.0

```

[2] ループ内演算数の増加

複数の演算パイプラインを有効活用するためには、種々のループ展開を行ってループ内の演算数を増やすとよいことが多い。このための方法として「最内側ループの解消」や「外側ループの展開」がある。

(a) 最内側ループの解消

次の例のように、最内側ループのループ長が小さな定数で、かつ外側にループがあるとき、最内側ループを解消して外側ループでベクトル化を行う。

```
DO 10 J=1,N
DO 20 I=1,3
20  A(I,J)=B(I,J)*C(I)
10  CONTINUE
      ↓
DO 10 J=1,N
A(1,J)=B(1,J)*C(1)
A(2,J)=B(2,J)*C(2)
10  A(3,J)=B(3,J)*C(3)
```

(b) 外側ループの展開

多重ループの場合、次の例のように外側ループに関して展開を行うと演算密度が高まり、ループに対するベクトル起動回数も減るため性能が向上する。

```
DO 10 J=1,256
DO 10 I=1,N
10  A(I,J)=B(I,J)*C(I,J)
      ↓
DO 10 J=1,256,4
DO 10 I=1,N
A(I,J)=B(I,J)*C(I,J)
A(I,J+1)=B(I,J+1)*C(I,J+1)
A(I,J+2)=B(I,J+2)*C(I,J+2)
10  A(I,J+3)=B(I,J+3)*C(I,J+3)
```

[3] ループ内ベクトル共通オペランドの増加

一般のプログラムをベクトル・プロセッサで実行する場合、性能を決めるのは主記憶に対する負荷によることが多い。次の例は、ループを展開すると、配列Cのロードが展開ループで共通であるため、全体のロード負荷量は展開数分の1に削減される。



```

DO 10 J=1,256
DO 10 I=1,N
10  A(I,J)=B(I,J)*C(I)
      ↓
DO 10 J=1,256,4
DO 10 I=1,N
A(I,J)=B(I,J)*C(I)
A(I,J+1)=B(I,J+1)*C(I)
A(I,J+2)=B(I,J+2)*C(I)
10  A(I,J+3)=B(I,J+3)*C(I)

```

#### [4] 条件文処理の効率化

条件文はマスクベクトルによりベクトル化できるが、THEN 節および ELSE 節も演算の実行は行われるため、使わない方法が考えられるのであれば、より望ましい。例えば次の例ではループ初回の特別処理を条件文を用いて実行している。このため、初回だけをループから分離すると、ループ内から条件文を除去することができ、効率的なベクトル化が可能となる。

```

DO 10 I=1,N
IF(I .EQ. 1) THEN
A(I)=0.0
ELSE
A(I)=B(I)+C(I)
END IF
10  CONTINUE
      ↓
A(1)=0.0
DO 10 I=2,N
A(I)=B(I)+C(I)
10  CONTINUE

```

この方法はループの初回、途中の点、最終回などを分離する方法で、「ループ端点展開」と呼ばれる <sup>47)</sup>。

## 第5章 要素間に相互依存性のある処理の高速化方式

### 5.1 総和型演算の高速化方式

ベクトル処理ではベクトル要素間の独立性によって処理の高速化を実現していることは以前に述べた通りであるが、実際には要素間に相互依存性のある処理もある。例えば

$$S = \sum A_i \times B_i$$

のような処理では一つのベクトル  $A$ 、 $B$  を乗算した後、その積の各要素をすべて加算する。 $A \times B$  は要素並列での演算が可能であるが、要素をすべて加算する処理は各演算器はそれぞれ独立に動作するため、そのままではできない。このため S-820 では各演算器ごとにまず乗算した後各加算器で部分和を計算し、各加算器の出力を集めて加算する専用の後処理演算器を新たに開発することによってこの問題の解決を図った。

例えば要素数が 80 の場合の内積演算を例に処理方法を説明する。

$$S = A_1 \times B_1 + A_2 \times B_2 + A_3 \times B_3 + \cdots + A_{80} \times B_{80}$$

図 5.1 に示すように、

- ・ 8 本のベクトル・ロード・パイプラインからベクトル・レジスタにベクトル  $A_i$  をロード。
- ・ 8 本のベクトル・ロード/ストア・パイプラインからベクトル・レジスタにベクトル  $B_i$  をロード。
- ・ 8 本の乗算・加算器で  $A_i$  と  $B_i$  の乗算および部分和を計算。

という順で実行することになる。これらの各ステップは 1 マシン・サイクルで実行できる。実行時間  $T$  は、パイプラインの本数が 8 本なので、

$$T = \alpha + (N/8) + \beta$$

となる。

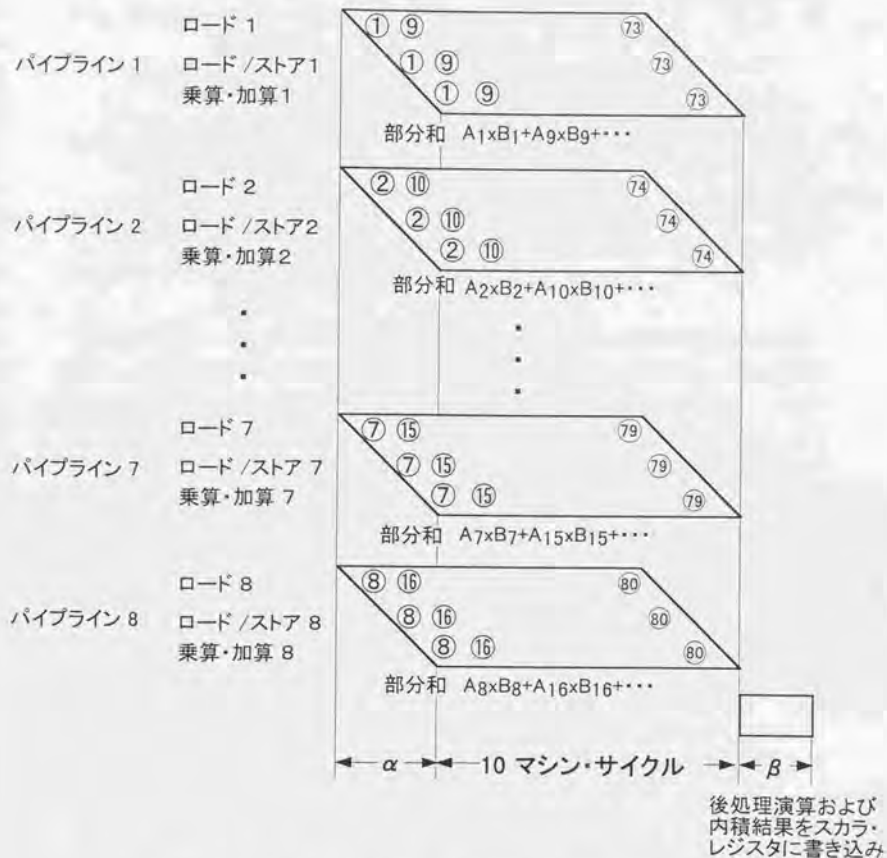


図5.1 要素並列処理(パイプライン)

( $\sum A_i \times B_i$  の例)

ここで、 $N$ は要素数、 $\alpha$ はパイプラインの始めに必要な前処理時間、 $\beta$ は後処理時間である。 $T$ の単位はマシン・サイクルで、この例では

$$10 \text{ サイクル} + \text{前処理時間 } \alpha + \text{後処理時間 } \beta$$

となる (図 5.1)。 $\alpha + \beta$ の時間は、配列の大きさに依存しない。

しかし、内積計算では各パイプラインの計算結果を一つにまとめる演算制御が必要である。後処理時間  $\beta$  がこれに当たる。この時間を短縮するために、S-820 ではベクトル要素間の演算を専用に行う後処理演算器を新たに開発した。

図 5.2 に示すように、8 本の乗算・加算パイプラインはそれぞれに並列に部分和を演算していく。最後の部分和は各パイプライン演算器内の結果レジスタに格納する。後処理演算器は 8 本の結果レジスタからその値を読み出し、加算を実行して内積結果をスカラ・レジスタに書き込む。

このように総和をとるタイプの演算は後処理演算器を用いて高速化を図ることが可能である。表 5.1 に後処理演算器を用いて高速化を図ったベクトル命令の種類を示す。

表 5.1 後処理演算器で高速化を図ったベクトル命令の種類

	ベクトル命令種類	演算動作
1	VSM(Vector Element Sum) 総和演算	$Z = \sum X_i$
2	VIP(Vector Inner Product) 内積演算	$Z = Z + \sum X_i \times Y_i$

内積演算は上記に述べた動作を行うが、総和演算の動作も同様に 8 本の加算パイプラインがそれぞれに並列に部分和を演算する。最後の部分和は各パイプライン演算器内の結果レジスタに格納され、後処理演算器が 8 本の結果レジスタからその値を読み出し、加算を実行して総和結果をスカラ・レジスタに書き込む。

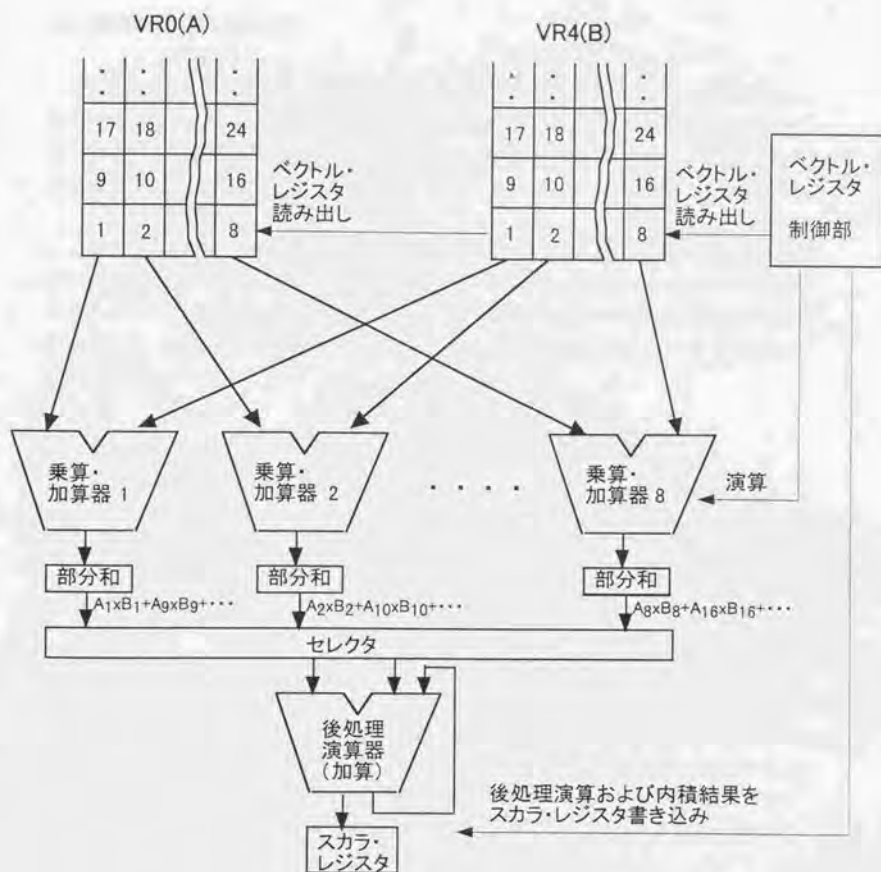


図5.2 要素並列処理(後処理演算器)

( $\sum A_i \times B_i$  の例)



## 5.2 巡回型演算の高速化方式

次に要素並列パイプライン方式では高速化が困難な要素間に相互依存性のある処理として巡回型の演算がある。巡回型の演算は表 5.3 に示すように、 $i$  番目の要素を演算するために、直前の  $i-1$  番目の要素を必要とする。このため、要素並列型演算パイプライン方式では高速化が難しい。

しかしこれらの巡回型演算は、科学技術計算の中では後退代入法として多く出現している。特に、大きな DO ループの中にも現れるため、ベクトル化率を上げるためにもベクトル命令として高速化を図ることが重要である。このため、S-820 では演算器に専用の制御回路を設け、高速化を図った。表 5.2 に専用制御演算器で高速化を図ったベクトル命令の種類を示す。

表 5.2 専用制御演算器で高速化を図ったベクトル命令の種類

	ベクトル命令種類	演算動作
1	VINC(Vector Element Increment) 逐次加算	$Z_n = SR + X_n$ for $n=0$ $Z_n = Z_{n-1} + X_n$ for $n>0$
2	VITR(First Order Iteration) 一次巡回演算	$Z_n = SR * Y_n + X_n$ for $n=0$ $Z_n = Z_{n-1} * Y_n + X_n$ for $n>0$

図 5.3 に S-820 における巡回型演算の高速化処理を図った演算ユニットの構成を示す。この演算ユニットは乗算器と加算器から構成されるが、乗算器の結果を加算器の入力に直接接続するパス（図 5.3 の S10）と、加算器の結果を自加算器の入力に直接接続するパス（図 5.3 の S11）を設け、巡回型演算のベクトル命令の種類に対応して乗算器の結果あるいは加算器の結果を、S10 あるいは S11 のパスを用いて加算器の入力に直接送る制御をおこなう点が特長である。VINC 命令（逐次加算）の場合はパス S11 を用い、VITR 命令（一次巡回演算）の場合はパス S10 およびパス S11 を用いて高速化を図った。

乗算器は 4 ステージからなる。第 1 ステージでは処理されるデータがレジスタ A とレジスタ B にセットされる。第 2 ステージではレジスタ A とレジスタ B の中間積がレジスタ M1 にセットされる。第 3 ステージでは同じく、中間積がレジスタ M2 にセットされる。第 4 ステージでは最終積が演算され結果がレジスタ M3 にセットされる。レジスタ M3 の最終積の内容はレジスタ R か、加算器の入力レジスタ D に送られる。

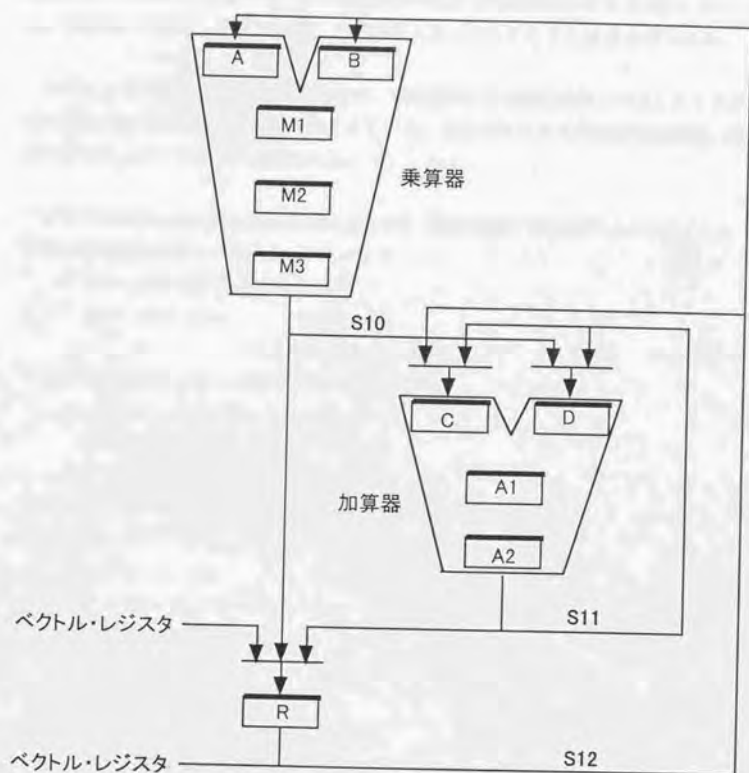


図5.3 演算ユニットの構成（巡回演算の高速化）

加算器は3ステージからなる。第1ステージでは処理されるデータがレジスタCとレジスタDにセットされる。第2ステージではレジスタCとレジスタDの中間和がレジスタA1にセットされる。第3ステージでは最終和が演算され結果がレジスタA2にセットされる。最終和の内容はレジスタRか、加算器の入力レジスタCまたはDに送られる。

レジスタRは乗算器による演算結果か、加算器による演算結果か、ベクトル・レジスタから送られてくる演算データが格納されている。またレジスタRの内容は加算器、乗算器またはベクトル・レジスタに送られる。

まずVINC(Vector Element Increment)命令(逐次加算)の動作について考える。  
VINC命令の演算式次に示す。

$$Z_n = SR + X_n \quad \text{for } n=0$$

$$Z_n = Z_{n-1} + X_n \quad \text{for } n>0$$

またこの式は  $n>2$  に対して次のように記述できる。

$$Z_n = Z_{n-2} + X_{n-1} + X_n$$

これらの式によりVINC命令の演算結果は次のように得られる。

$$\begin{aligned} Z_2 &= Z_0 + X_1 + X_2 \\ &= SR + X_0 + X_1 + X_2 \\ &= (SR + X_1) + (X_0 + X_2) \end{aligned}$$

$$\begin{aligned} Z_3 &= Z_1 + X_2 + X_3 \\ &= SR + X_0 + X_1 + X_2 + X_3 \\ &= (SR + X_1 + X_3) + (X_0 + X_2) \end{aligned}$$

$$\begin{aligned} Z_4 &= Z_1 + X_1 + X_2 + X_3 + X_4 \\ &= SR + X_0 + X_1 + X_2 + X_3 + X_4 \\ &= (SR + X_1 + X_3) + (X_0 + X_2 + X_4) \end{aligned}$$

⋮  
⋮  
⋮

この式に示すように、 $Z_n$  は奇数番の部分  $SR + X_1 + X_3 + \dots$  と偶数番の部分  $X_0 + X_2 + X_4 + \dots$  から構成することができる。従って、 $Z_n$  の処理は奇数番の部分と偶数番の部分とをそれぞれ独立に演算していくことにより高速化することが可能になる。

図 5.4 に S-820 における VINC 命令の高速処理を示す。図 5.4 は図 5.3 における各演算器のステージの内容を時間 (マシン・サイクル) 軸毎に示したものである。この図において、演算データ  $X_i$  は 2 サイクル毎にベクトル・レジスタから供給され、レジスタ R にセットされる。すなわち、 $t = 5, 7, 9, \dots$  の時である。

各奇数サイクルには、加算器の入力レジスタ C に奇数番の部分  $SR$  が、レジスタ D には偶数番の部分  $X_0 + X_2 + \dots$  がセットされる。従って  $t = 8$  の時から各 2 サイクル毎にレジスタ R に各要素の演算結果が得られる。

図 5.5 に図 5.4 と同じ演算器構成での逐次処理のステージを示す。この図に示すように、 $Z_{n+1}$  の演算は  $Z_n$  の結果が得られてから始まる。このため各 3 サイクル毎にレジスタ R に各要素の演算結果が得られる。

このような逐次巡回処理の分解方式 (reduction) については、ソフトウェアによるものは "the recursive doubling method" として報告されているが<sup>53)</sup>、ハードウェアによる高速化方式および実現方式に関してはこの報告が初めてである<sup>54)</sup>。

t=1		t=2		t=3		t=4		t=5		t=6		t=7		t=8	
C	D			SR		SR	X1	SR	X0	X2	X0	SR+X1	X0	SR+X1	X3
A1								SR+X1		Z0=SR+X0		X2+X0		Z1=SR+X1+X0	
A2										SR+X1		Z0		X2+X0	
R		SR		X1		X0		X2				X3		Z0	

t=9		t=10		t=11		t=12		t=13		t=14		t=15		t=16	
SR+X1	X2+X0	X4	X2+X0	SR+X1+X3	X2+X0	SR+X1+X3	X5	SR+X1+X3	X2+X0+X4	X6	X2+X0+X4	X2+X0+X4		X7	
SR+X1+X3		Z2		X2+X0+X4		Z3		SR+X1+X3+X5		Z4		X2+X0+X4+X6		Z5	
Z1		SR+X1+X3		Z2		X2+X0+X4		Z3		SR+X1+X3+X5		Z4		X2+X0+X4+X6	
X4	Z1			X5		Z2		X6		Z3		X7		Z4	

t=17		t=18		t=19		t=20		t=21		t=22		t=23		t=24	
SR+X1+X3+X5+X7		X8				X9				X10				X11	
Z5		Z6		Z6		Z7		Z7		Z8		Z8		Z9	
		SR+X1+X3+X5+X7													
X8	Z5			X9		Z6		X10		Z7		X11		Z8	

図5.4 逐次加算演算の高速化 (VINC)



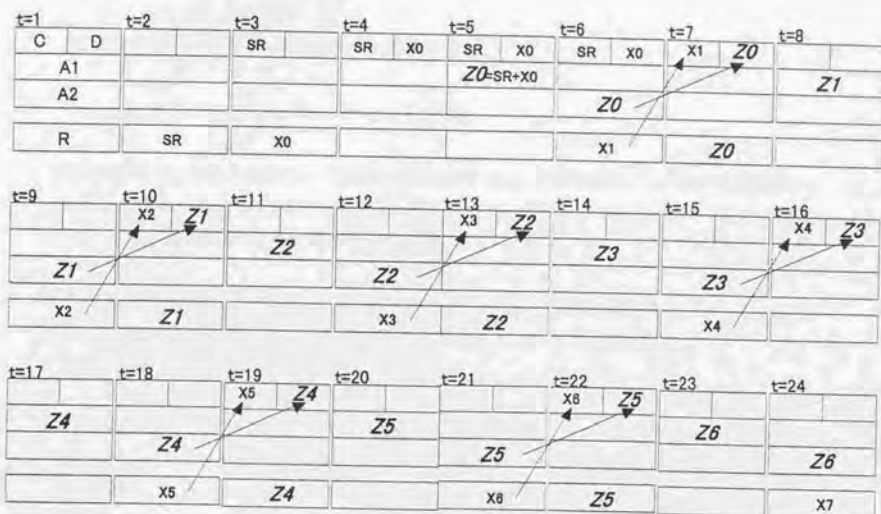


図5.5 逐次加算演算の通常処理 (VINC)

次に VITR(First Order Iteration)命令 (一次巡回演算) の動作について考える。  
VITR 命令の演算式次に示す。

$$\begin{aligned} Z_n &= SR * Y_n + X_n && \text{for } n=0 \\ Z_n &= Z_{n-1} * Y_n + X_n && \text{for } n>0 \end{aligned}$$

またこの式は  $n>2$  に対して次のように展開できる。

$$Z_n = Z_{n-2} \times Y_{n-1} \times Y_n + X_{n-1} \times Y_n + X_n$$

この式から  $Z_n$  を得るために、直前の演算結果の  $Z_{n-1}$  からではなく 2 つ前の演算結果の  $Z_{n-2}$  から求めることが可能である。しかも  $Y_{n-1} \times Y_n$  の演算と  $X_{n-1} \times Y_n + X_n$  の演算は  $Z_{n-2}$  の演算とは独立に処理することができる。

このことを利用した S-820 における VITR 命令の高速処理を図 5.6 に示す。この図は VINC 命令の例と同様に図 5.3 における各演算器のステージの内容を時間 (マシン・サイクル) 軸毎に示したものである。

図 5.6 において、 $Z_3$  の演算は  $Z_1$  の演算結果が得られる  $t=14$  より前の  $t=12$  より開始される。すなわち、 $Y_2 \times Y_3$  の演算は  $t=12$  より開始され、 $X_2 \times Y_3 + X_3$  の演算は  $t=13$  より開始される。これらの演算はいずれも  $Z_1$  とは独立に演算可能である。

また  $Z_1$  を使用した演算、すなわち  $Z_{n-2} \times Y_{n-1} \times Y_n$  の演算は、 $Z_1$  の結果が得られた  $t=14$  の直後の  $t=16$  より開始される。これは  $Z_2$  の結果が得られる  $t=22$  より 6 サイクルも前である。

この図に図に示すように、VITR 命令の処理速度は 9 サイクルに 2 演算結果である。  
すなわち、

- $Z_0$  は  $t=13$  に結果が得られ、
- $Z_1$  は  $t=14$  に結果が得られ、
- $Z_2$  は  $t=22$  に結果が得られ、
- $Z_3$  は  $t=23$  に結果が得られ、

である。

$n$  が偶数の場合、 $Y_{n-1} \times Y_n$  と  $X_{n-1} \times Y_n + X_n$  の演算は  $Z_n$  が得られる前に前もって演算が可能のため、 $Z_{n+1}$  は  $Z_n$  の結果が得られる次のサイクルで得られる。言い換えれば  $Z_n$  と  $Z_{n+1}$  の演算が並列に処理することができる。

図 5.7 に図 5.6 と同じ演算器構成での逐次処理のステージを示す。この図に示すように、 $Z_{n+1}$  の演算は  $Z_n$  の結果が得られてから開始される。このため各 8 サイクル毎にレジスタ  $R$  に各要素の演算結果が得られる。

t=1		t=2		t=3		t=4		t=5		t=6		t=7		t=8	
A	B	Y0		Y0	Y1	X0	Y1	Y0		Y0		Y0	SR	Y0*Y1	SR
M1						Y0*Y1		X0*Y1						Y0*SR	
M2								Y0*Y1		X0*Y1					
M3										Y0*Y1		X0*Y1			
C D															
A1															X0*Y1
A2															
Y0		Y1		X0		Y0				SR		Y0*Y1		X1	

t=9		t=10		t=11		t=12		t=13		t=14		t=15		t=16		X3
Y0*Y1*SR				Y2		Y2	Y3	X2	Y3	Y2		Y2		Y2		Z1
Y0*SR		Y0*Y1*SR						Y2*Y3		X2*Y3		Y2*Y3		X2*Y3		
		Y0*SR		Y0*Y1*SR								X2*Y3		Y2*Y3		X2*Y3
X1	X0*Y1	X0		X0	Y0*SR	X0*Y1-X1	Y0*Y1*SR									
		X0*Y1+X1				Z0		Z1								
				X0*Y1+X1				Z0		Z1						
X0		Y2		Y3		X2		Y2		Z0		Z1		Y2*Y3		

t=17		t=18		t=19		t=20		t=21		t=22		t=23		t=24	
Y2*Y3	Z1					Y4		Y4	Y5	X4	Y5	Y4		Y4	
Z1*Y2		Z1*Y2*Y3								Y4*Y5		X4*Y5		X4*Y5	
		Z1*Y2		Z1*Y2*Y3								Y4*Y5		X4*Y5	
				Z1*Y2		Z1*Y2*Y3								Y4*Y5	
	X2*Y3	X3	X2*Y3	X2		X2	Z1*Y2	X2*Y3+X3	Z1*Y2*Y3						
				X2*Y3+X3				Z2		Z3					
						X2*Y3+X3				Z2		Z3			
X3		X2		Y4		Y5		X4		Y4		Z2		Z3	

図5.6 一次巡回演算の高速化 (VITR)

t=1		t=2		t=3		t=4		t=5		t=6		t=7		t=8	
A	B	Y0		Y0	SR										
M1						SR*Y0									
M2								SR*Y0							
M3										SR*Y0					
C	D					X0		X0		X0		X0	SR*Y0		
A1															
A2														Z0	
Y0		SR		X0											

t=9		t=10		t=11		t=12		t=13		t=14		t=15		t=16	
				Y1	Z0										
						Z0*Y1									
								Z0*Y1							
										Z0*Y1				X2*Y3	
												X1	Z0*Y1		
Z0														Z1	
Y1		Z0								X1					

t=17		t=18		t=19		t=20		t=21		t=22		t=23		t=24	
		Y2		Y2	Z1										
						Z1*Y2									
								Z1*Y2							
										Z1*Y2					
												X2	Z1*Y2		
Z1														Z2	
Y2		Z1								X2					

図5.7 一次巡回演算の通常処理 (VITR)



### 5.3 コンパイラ等による高速処理方式

4.5で述べたように、この論文では筆者が主設計者として実現したアーキテクチャおよびハードウェアの論理方式に関して述べているが、性能向上の点から関連するコンパイラ技術に関しても記述する。逐次加算演算および巡回型演算に関しては、これまでは高速化が不十分かあるいはコンパイラによるベクトル化の範囲が狭かった。そこで次に述べる性能向上対策を検討した。ただしコンパイラによる高速化項目については、実現の可否を決定したが、具体的な実現はコンパイラ担当者が行った。

#### 5.3.1 逐次加算演算の特殊処理

次のような FORTRAN の DO ループは出現頻度が高く、これを高速に処理するために、数列の生成を高速にベクトル処理で行うことにした。

```
DO 10 I=1,100  
10 A(I)=I
```

コンパイラではこのオブジェクトとして、VINC 命令を出す。通常の VINC 命令の処理は 2 サイクル・ピッチで結果を得るが、VINC 命令の  $X_n$  にスカラデータが指定されたときには、要素並列でパイプライン処理する方式を開発した。これは、スカラデータを最初にパイプラインに設定し、要素並列処理型パイプラインを構成する各物理パイプラインごとにカウントアップする値を管理して加算処理を行うことにより実現した。これにより、1 サイクル・ピッチで 8 つの結果を得ることができる。

#### 5.3.2 巡回型演算を含む高速化

巡回型演算を含む高速化に関しては、ハードウェアの高速化のみならず、コンパイラからのプログラム変換による高速化が大きい<sup>34)</sup>。例えば、リバモア 14 ループの中に現れる一般的な巡回型演算（三角行列の後退代入）を含むプログラムは FORTRAN で次のように記述されているが、従来のコンパイラ技術では複数の文にわたる一次巡回型依存関係が解析できずベクトル化が出来なかった。

```
DO 10 J=2,N,3  
A(J)=B(J)*A(J-1)+C(J)          S1  
A(J+1)=B(J+1)*A(J)+C(J+1)      S2  
A(J+2)=B(J+2)*A(J+1)+C(J+2)    S3  
10 CONTINUE
```

このループにおいて文 S1, S2, S3 は明らかに配列 A に関するフロー依存により強い連結を構成している。さらに文 S3 から S1 への依存はループ 2 回遅延フローであることから、ループの繰り返し 1 回前の値を使用する一次巡回型関係であることがわかる。

このような複数の文にわたる一次巡回型関係は、次の条件が成立する時、一次巡回型演算をプログラム変換により 1 つの文の内に閉じさせることができる。

[条件]

- ・強い連結成分がフロー依存のみで構成される。
- ・フロー依存の種類として、1 つのデータ依存グラフの属性がループ 2 回依存のみ、他のデータ依存グラフの属性はループ独立依存のみである。

以下にこれを適用したプログラム変換過程を示す。

[プログラム変換 1]

```

DO 10 J=2,N,3
  A(J)=B(J)*A(J-1)+C(J)                      S1
  A(J+1)=B(J+1)*A(J)+C(J+1)                  S2
  A(J+2)=B(J+2)*(B(J+1)*(B(J)*A(J-1)+C(J)) + C(J+1))+C(J+2)  S3
10  CONTINUE

```

[プログラム変換 2]

```

DO 10 J=2,N,3
  E= B(J+2)*B(J+1)*B(J)
  F= B(J+2)*B(J+1)*C(J)+ B(J+2)*C(J+1)+ C(J+2)
  A(J+2)=E*A(J-1)+F                            一次巡回型演算命令適用
  A(J)=B(J)*A(J-1)+C(J)
  A(J+1)=B(J+1)*A(J)+C(J+1)
10  CONTINUE

```

[プログラム変換 2] においては、一次巡回型演算部が 1 つの文の内に閉じており、一次巡回型演算のベクトル命令が適用可能となった。

リバモア 14 ループの例では、kernel 5, 6 がこのタイプに属し、新たにベクトル化が可能になり高速化が実現した。

## 第6章 ベクトル処理の分割起動方式と記憶制御方式

### 6.1 ベクトル処理の分割起動方式

ベクトル演算を開始するに当たっては、ベクトル・アドレスや処理ベクトル要素数の指定などを行う必要がある。これらの処理を総じてベクトル準備処理とよぶ。前述のようにベクトル準備処理はスカラ処理ユニットで実行される。この処理が完了すると、起動命令 (EXVP: EXecute Vector Processing 命令) を発行してベクトル処理ユニットに対し起動をかけ、ベクトル処理を開始する (図 6.1 参照)。

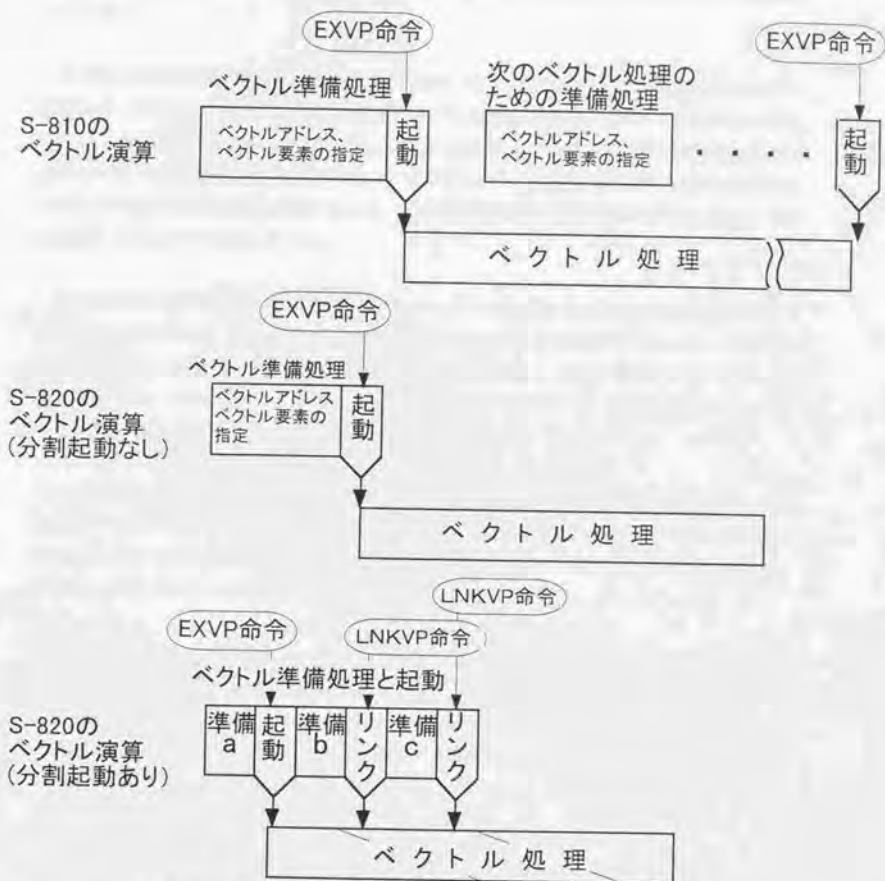
S-810 では、あるベクトル演算についてのすべてのベクトル準備処理が完了してからベクトル処理を起動していた。この後スカラ処理ユニットでは、継続するスカラ処理または次のベクトル処理のためのベクトル準備処理を実行させるように考えていた。こうすることによってスカラ処理ユニットとベクトル処理ユニットの並列実行を実現させ、システム全体の実効性能を向上させようとした (図 6.1 S-810 の例を参照)。

しかし当初想定したほどベクトル処理ユニットとスカラ処理ユニットの並列動作が行われず、従ってベクトル準備処理をかくことができず、とくにベクトル要素数が少ない場合に期待したほど実効性能を向上できなかった。この原因としては、ベクトル処理が完了し、その結果を用いて若干スカラ処理をしてから再びベクトル処理を行うようなケースが少なくなかったことが挙げられる。

一方、S-820 ではベクトル処理そのものの性能向上に加え、要素並列型演算パイプライン方式の開発により S-810 に比べて実効性能が向上しているので、ベクトル処理時間が大幅に短縮している。しかしベクトル準備処理時間は、そのままではスカラ処理ユニットのマシンサイクルの短縮による性能向上にとどまってしまう。

ベクトル要素数 100 で  $A = B \times C + D$  のベクトル演算を実行する場合の例を図 6.2 に示す。この実行時間は次のようになる (カッコ内は合計に占める割合)。

	ベクトル準備処理	ベクトル処理	合 計
S-810	0.3 $\mu$ s (11%)	2.5 $\mu$ s (89%)	2.8 $\mu$ s
S-820	0.2 $\mu$ s (20%)	0.8 $\mu$ s (80%)	1.0 $\mu$ s



[準備処理命令数に着目した分割方式を開発]

図6.1 ベクトル処理の分割起動方式



すなわち、ベクトル準備処理時間の全体に占める割合が、S-820 では S-810 の約 2 倍になってしまう。

ベクトル準備処理時間の短縮のために、S-820 ではベクトル演算の準備処理とベクトル処理をオーバーラップさせることにした。図 6.1 の下段にこの様子を示す。すなわち、1つのベクトル処理をいくつかに分割し（この例は a,b,c の 3 分割の例）、最初の分割部分に相当するベクトル準備処理が完了すると、まずそのベクトル処理を起動する。次の分割されたベクトル処理の準備処理が完了すると、引き続き次のベクトル処理が継続できるように起動し、以下これを繰返していく。

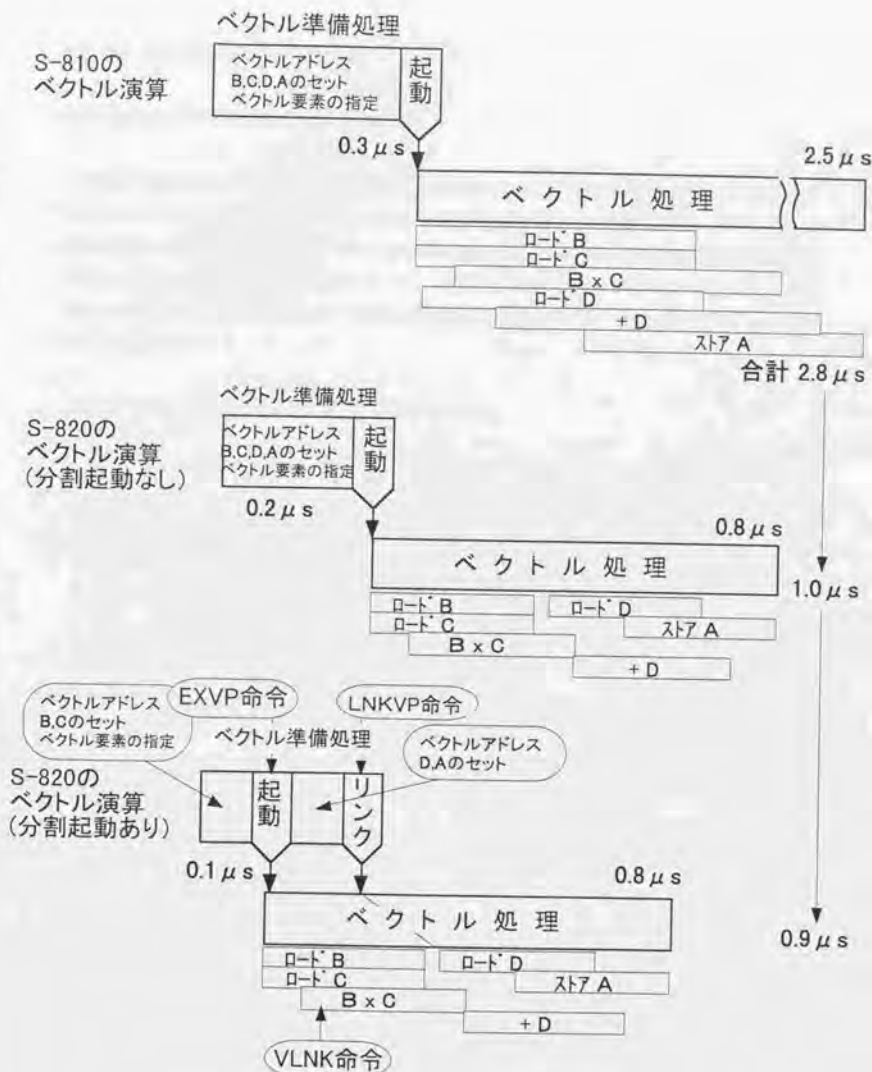
この具体的な例を図 6.2 の下段に示す。このベクトル処理は  $A = B \times C + D$  の例なので 2 分割起動の例である。最初のベクトル準備処理ではまず B および C のベクトル・アドレスをセットし次に処理ベクトル要素数の指定をセットする。この時点でまずそのベクトル処理を起動する。次に D および A のベクトル・アドレスをセットし、引き続きその後のベクトル処理が継続できるように起動する。

このようにベクトル処理を分割して起動することにより、ベクトル準備処理とベクトル処理をオーバーラップさせることができ、見かけ上の準備処理のオーバーヘッドをこの場合は 1/2 の  $0.1\mu s$  に短縮できる。本例は小規模なベクトル演算であるが、さらに規模の大きなベクトル演算ではオーバーヘッド削減の効果はさらに大きい。

この分割起動方式を実現するに当たっては、ベクトル処理起動命令（前述の EXVP 命令）に加えて LNKVP (Link Vector Processor 命令) を新設し、ベクトル処理実行中でもベクトル処理ユニット側で VLNK (Vector Link) 命令が実行済であれば、引き続きベクトル処理を継続できるようにした（EXVP 命令を実行するためには、ベクトル処理ユニットは何らの動作も行っていない状態になっていなければならない）。これによって演算器等の動作に空きができることを防いだ。

このようなベクトル処理起動命令によって、ベクトル処理を起動させるやり方はスカラ処理の命令コードとベクトル処理の命令コードを完全に分けていることに起因している。日立以外のベクトルプロセッサ方式は命令コードを混在している (Cray、富士通、日電) 場合が多いので、上記のようなアーキテクチャは日立特有のものである。しかしいづれにしてもベクトル処理の起動および終了をスカラ処理側が管理する（アーキテクチャおよびそのハードの制御）という考え方は同様である。





[準備処理命令数に着目した分割方式を開発]

図6.2 ベクトル処理の分割起動方式

( $A = B \times C + D$ 、100要素の例)

## 6.2 ベクトル処理の主記憶制御方式

### 6.2.1 S-820 における高速主記憶制御方式

いままでは主として、ベクトル処理の演算制御方式および命令制御方式について記述してきたが、ここでは主としてベクトル処理の主記憶制御方式について述べる。ベクトル処理を高速に処理するためには、ベクトル命令の効率良い制御方式や演算制御方式が必須であるが、演算データの主記憶からベクトル・レジスタへの転送およびベクトル・レジスタから主記憶への転送を効率良くおこなう主記憶制御方式は、ベクトル処理性能にとってきわめて重要である。

図 6.3 に S-820 で実現した主記憶制御方式を示す。

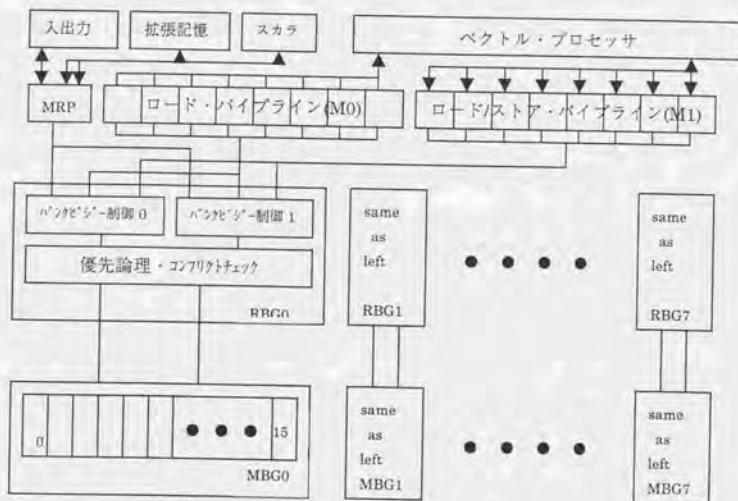


図 6.3 HITAC S-820 の主記憶制御構成図

S-820 の主記憶は 128 バンクのインターリーブから構成される。まず 8 個の独立にアクセス可能なバンク・グループ (図 6.3 の RBG0~RBG7) があり、それぞれのバンク・グループは 16 個のバンクを持っており、それらが 8 個ある (図 6.3 の MBG0~MBG7)。

主記憶へのアドレスは31ビットであるが、アドレス・フィールドは次の4フィールドに分けられる。

- ・ビット 1-21 : 上位アドレス
- ・ビット 22-25 : バンク・アドレス
- ・ビット 26-28 : バンク・グループ・アドレス
- ・ビット 22-25 : バイト・アドレス

主記憶へのアクセスは、入出力・拡張記憶・スカラプロセッサからのアクセスが Memory Requester P (MRP) でひとまとめにスタックされる。ベクトルプロセッサからのアクセスは Memory Requester 0 (M0) と Memory Requester 1 (M1) とでスタックされる。M0 はロード・パイプラインに対応し、M1 はロード/ストア・パイプラインに対応している。M0 と M1 はそれぞれ独立なベクトル命令によって実行される。

ベクトル処理は要素並列型演算・パイプライン方式により処理されているので、M0 および M1 はそれぞれ8要素単位でアクセスが行われる。また主記憶へのアクセスは、8B 単位に処理される。このため、また主記憶への同時アクセスは、入出力・拡張記憶・スカラプロセッサからのアクセスが1つ、M0 からのアクセスが8つ、M1 からのアクセスが8つの合計 17 個の可能性ある。アドレスのビット 26-28 で指定されたバンク・グループ8個 (RBG0-7) が独立にかつ並列にこれらのアクセスを処理する。

それぞれのバンク・グループは、バンク・ビジー制御 0 とバンク・ビジー制御 1 の2つのバンク・ビジー制御回路を持ち、最大 17 個の主記憶への同時アクセスを1クロックに2つ処理する。バンク・ビジーでないアクセスが優先論理・コンフリクト・チェック回路に入る。互いに同じバンクを指すアクセスの場合は、優先がとられるが、バンクが異なる場合は同時にメモリ・バンクにアクセスされる。

バンク・グループ8個 (RBG0-7) からの主記憶アクセスは、対応するバンク (MBG0-7) へ送られる。バンク番号は、アドレスのビット 23-25 で指定される。

S-820 の主記憶制御部分は1クロック = 8ns であるので、この主記憶のピーク・スルーputは

$$2 \text{ (主記憶7ヶ所)} * 8 \text{ B} / 8 \text{ ns} = 16 \text{ GB/sec}$$

となる。

## 6.2.2 セクション番号アサイン方式による高速化

次に主記憶のスループットを高めるための方式について述べる。その一つはバンク・グループ制御における高速化方式である。一般に主記憶アクセスが同時に発生する場合、1クロックでバンク・ビジーとコンフリクト・チェックおよび優先付けを行うのは難しく、数クロックを要する。この制御をステージと考えて、アクセスが連続した場合、逐次処理を行ったケースの処理を図6.4に示す。この図ではステージ方式数を4としている。この結果、逐次処理では主記憶へのアクセスは4クロック・ビッチとなってしまふ。このため、S-820では1クロック・ビッチアクセスを実現するために、セクション番号アサイン方式を考えた。

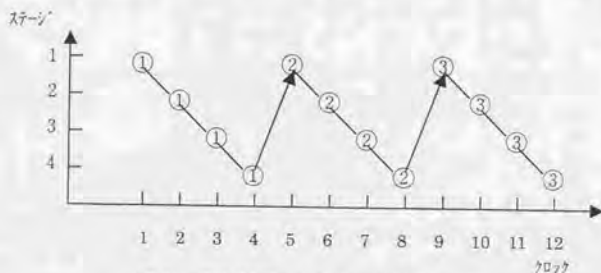


図 6.4 逐次処理による主記憶制御



図 6.5 セクション番号アサイン方式による主記憶制御

まずベクトルを4々のセクションに分ける。1つのセクションには8ベクトル要素が含まれている。セクション番号のアサイン方式を図6.6に示す。例えば、ベクトル要素0-7はセクション番号1をアサインし、ベクトル要素8-15はセクション番号2をアサインする。このようにしていくと、ベクトル要素32-39に同じセクション番号1がアサインされる。



次に主記憶へのアクセスは、セクション番号が異なる場合は並列に実行できるようにする。図 6.6 の方式では、ベクトル要素 0-31 の処理が並列に処理される。

セクション番号	ベクトル要素並列番号							
1	0	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14	15
3	16	17	18	19	20	21	22	23
4	24	25	26	27	28	29	30	31
1	32	33	34	35	36	37	38	39
2	40	41	42	43	44	45	46	47
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
ロード・パイプライン (M1)の対応リクエスト	S00	S01	S02	S03	S04	S05	S06	S07

図 6.6 セクション番号アサイン方式

このセクション番号アサイン方式の処理の様子を図 6.5 に示す。この図で主記憶へのアクセス 1, 2, 3, 4 は、それぞれセクション番号 1, 2, 3, 4 に対応する。これらの 4 つの主記憶へのアクセスは互いに独立に並列に処理される。5 番目のアクセスはセクション番号 1 に相当するので、1 番目のアクセスが主記憶のバンクへ送られることを確認(ステージ 4)してから開始される。この方式を採用することにより、主記憶へのアクセスが 1 クロック・ピッチで実現できた。

### 6.2.3 バンク・グループ番号モディファイ方式による高速化

ここでは、主記憶スループットを低下させる要因のうち、バンク・グループ・コンフリクトについて述べる。一般に、主記憶に対するアクセス要求が増加するのに従ってその要求を満たすための論理量が増加していく。しかし論理量の増加はコストの点で好ましくないで、主記憶に対するアクセス要求が特定の論理に集中しない様にする工夫が必要になる。

例えば図 6.6 において、ベクトル要素並列番号に対応して、ロード・パイプライン(M1)の対応リクエストがアサインされる。この図においてベクトル要素並列番号 0, 8, 16, 24, 32, 40 を処理するリクエストは S00 である。ここでベクトル・ストライドが 1 (連続アド



レス) の場合のベクトル・ロード命令の処理を考える。図 6.7 はバンク・グループ番号モディファイをおこなわない場合の主記憶アクセスの処理を示す。

この図において各ボックスの中は、8B 単位の主記憶アドレスとロード・パイプライン(M1)の対応リクエスト(例えば S00)を示す。横軸の内容はバンク・グループ番号を示し、縦軸の内容はそれぞれのバンク・グループ内におけるバンク番号を示す。

バンク・グループ No.	0	1	2	3	4	5	6	7
0	0 S00	1 S01	2 S02	3 S03	4 S04	5 S05	6 S06	7 S07
1	8 S00	9 S01	10 S02	11 S03	12 S04	13 S05	14 S06	15 S07
2	16 S00	17 S01	18 S02	19 S03	20 S04	21 S05	22 S06	23 S07
3	24 S00	25 S01	26 S02	27 S03	28 S04	29 S05	30 S06	31 S07
4	32 S00	33 S01	34 S02	35 S03	36 S04	37 S05	38 S06	39 S07
5	40 S00	41 S01	42 S02	43 S03	44 S04	45 S05	46 S06	47 S07
6	48 S00	49 S01	50 S02	51 S03	52 S04	53 S05	54 S06	55 S07
7	56 S00	57 S01	58 S02	59 S03	60 S04	61 S05	62 S06	63 S07
8	64 S00	65 S01	66 S02	67 S03	68 S04	69 S05	70 S06	71 S07

図 6.7 バンク・グループ番号モディファイをおこなわない場合の主記憶アクセス

まず最初に図 6.7 の第一行の 8 つの主記憶アクセスが、バンク・グループの制御に送られる。続いて第二行の 8 つの主記憶アクセスが、送られる。さらに第三行、第四行の 8 つの主記憶アクセスが続く。このようにすると同じリクエスト(例えば S00)が連続して同じバンク・グループの制御に送られる。図 6.7 のハッチング部分はこのことを示している。この場合、主記憶アクセスのリクエストを処理するバンク・グループの制御側では、同じリクエスト(例えば S00)からの連続リクエストとなるので、リクエストを処理するスタックが不足して処理性能が低下する。

この問題を解決するために、S-820 では「バンク・グループ番号モディファイ方式」を開発した。この方式においてバンク・グループ番号は次のように表現される。

$$\text{バンク・グループ番号} = [\text{バンク・グループ番号} - \text{バンク・アドレス下ビット}] \bmod 8$$

図 6.8 にこのバンク・グループ番号モディファイをおこなった場合の主記憶アクセス処理を示す。これも図 6.7 と同じベクトル・ストライドが 1 (連続アドレス) の場合のベクト

ル・ロード命令の処理である。図 6.8 から判る様に、同じリクエスト（例えば S00：ハッチング部分）が連続して同じバンク・グループの制御に送られることが回避されている。これは、図 6.7 においては、バンク・グループ 0 が処理するリクエストが毎サイクル S00 であったのに対し、図 6.8 では S00, S01, S02, S03... と毎サイクルリクエストが異なっている。このようにして、バンク・グループの制御のコンフリクトを避けるとともに、主記憶スループットを高めることができた。

バンク No. \ バンク・グループ No.	0	1	2	3	4	5	6	7
0	0 S00	1 S01	2 S02	3 S03	4 S04	5 S05	6 S06	7 S07
1	9 S01	10 S02	11 S03	12 S04	13 S05	14 S06	15 S07	8 S00
2	18 S02	19 S03	20 S04	21 S05	22 S06	23 S07	16 S00	17 S01
3	27 S03	28 S04	29 S05	30 S06	31 S07	24 S00	25 S01	26 S02
4	36 S04	37 S05	38 S06	39 S07	32 S00	33 S01	34 S02	35 S03
5	45 S05	46 S06	47 S07	40 S00	41 S01	42 S02	43 S03	44 S04
6	54 S06	55 S07	48 S00	49 S01	50 S02	51 S03	52 S04	53 S05
7	63 S07	56 S00	57 S01	58 S02	59 S03	60 S04	61 S05	62 S06
8	64 S00	65 S01	66 S02	67 S03	68 S04	69 S05	70 S06	71 S07

図 6.8 バンク・グループ番号モディファイをおこなった場合の主記憶アクセス

## 第7章 実現した方式の性能評価

### 7.1 要素並列型演算パイプライン方式の性能評価

演算器の利用率について、先に示したリバモアループ14ループ中のベクトル化された8ループについて解析した結果を表7.1と図7.1に示す。表7.1は8ループに対して、ループ中でのロード、ストア、加算、乗算の回数、S-820、S-810でのロード/ストア回路、加算器、乗算器の利用率を示したものである。また図7.1はロード/ストア回路、加算器、乗算器の利用率の平均値をS-820とS-810について比較したものである。

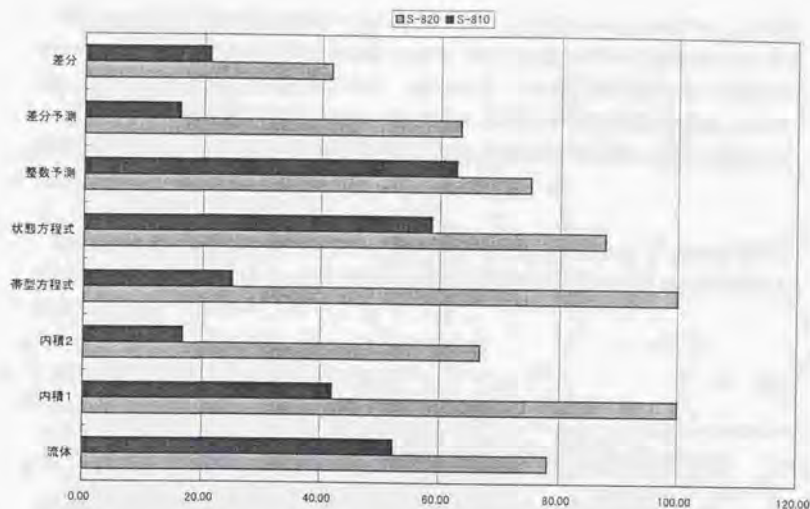
演算器およびロード/ストア回路の平均利用率は

	S-810	S-820
ロード/ストア回路	0.36	0.82
加 算 器	0.38	0.75
乗 算 器	0.38	0.73

に示すようにS-820の平均利用率はS-810に比較して約2倍に向上している。要素並列型演算パイプライン方式は演算器の利用効率を向上させることによって実効性能向上に大きく貢献しているといえる。

これを個別のループで比較すると、図7.1・表7.1において整数予測や状態方程式のループのように、ループ内の演算回数やメモリアクセスの多いものについては、S-810/S820の利用率の差は少ない。しかし内積2や帯型方程式のループのように、ループ内の演算回数やメモリアクセスの少ないものについては、S-810/S820の利用率の差は大きい。要素並列型演算パイプライン方式は、ループ内の演算回数やメモリアクセスの少ないものについて多大の効果がある。

なお、ロード/ストア：加算：乗算の実行回数について8ループの平均をとると、7.8：4.4：3.3となる。整数比に換算するとこれはほぼ2：1：1とみることができ、これが、ロード/ストア回路、加算器、乗算器（組）数をS-820では2：1：1、S-810では8：4：4とした理由となっている。



演算器およびロード・ストア回路利用率 (%)

図 7.1 S-810 と S820 の演算器およびロード・ストア回路利用率比較

表 7.1 S-820/S-810 の演算器およびロード・ストア回路利用率

(リバモアループ 14 ループ中の 8 ループ)

loop 番号	loop 名	loop 中の回数				S-820 の利用率			S-810 の利用率		
		ロード・ストア	ストア	加算	乗算	ロード・ストア	加算	乗算	ロード・ストア	加算	乗算
		2 組 16 個	1 組 8 個	1 組 8 個	1 組 8 個	2 組 16 個	1 組 8 個	1 組 8 個	2 組 16 個	1 組 8 個	1 組 8 個
1	流体	3	1	2	3	0.67	0.67	1.0	0.5	0.5	0.57
2	内積 1	10	0	5	5	1.0	1.0	1.0	0.42	0.42	0.42
3	内積 2	2	0	0	1	1.0	0	1.0	0.25	0	0.25
4	帯型方程式	2	0	1	1	1.0	1.0	1.0	0.25	0.25	0.25
7	状態方程式	9	1	8	8	0.63	1.0	1.0	0.42	0.67	0.67
9	整数予測	10	1	9	8	0.55	0.9	0.8	0.46	0.75	0.67
10	差分予測	10	10	9	0	1.0	0.9	0	0.25	0.23	0
12	差分	2	1	1	0	0.75	0.5	0	0.38	0.25	0
平均		6	1.8	4.4	3.3	0.82	0.75	0.73	0.36	0.38	0.38
(比率)		7.8				(2.3)	(2.0)	(1.9)	(1)	(1)	(1)



次に方式面から S-810 と S-820 の性能の比較を行う。表 7.1 にも記したように、S-820 では S-810 と比較してロード/ストア回路、加算器、乗算器の数をそれぞれ 2 倍にしてある。例えばロード/ストア回路は 8 → 16 個、加算器は 4 → 8 個、乗算器は 4 → 8 個である。S-810 ではこれらを単独で使用していたのに対し、S-820 では要素並列型演算パイプライン方式のため、ロード/ストア回路は 8 個を 1 組に、加算器、乗算器も 8 個を 1 組にして制御を行っている。

これらの方式の効果を比較するため、性能をクロック比で比較した。この様子を図 7.2 に示す。図 7.2 において S-810 での性能を 1 とし、S-820 の各種方式を S-810 に対する倍率で示した。

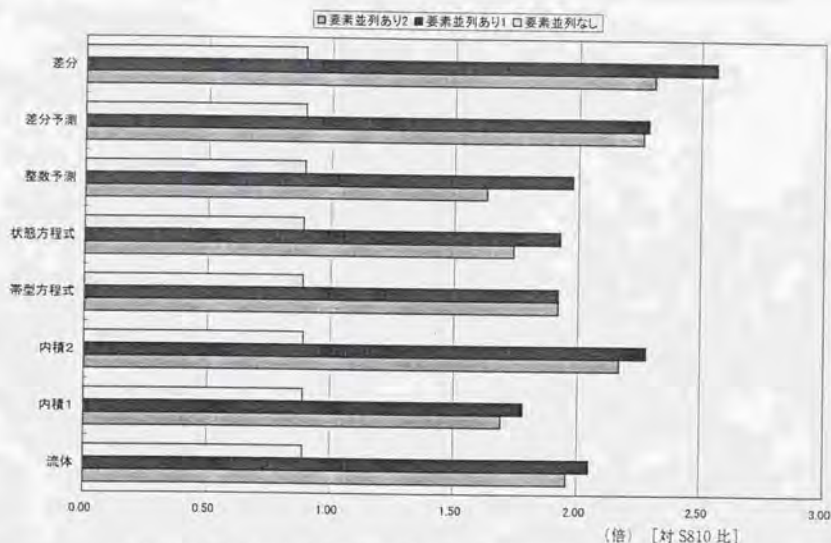


図 7.2 要素並列型演算パイプライン方式の方式評価

まず要素並列型演算パイプライン方式なしの場合の、S-810 と S-820 のクロックの比較は、本来ならば 1 倍となるべきであるが、実際は 0.9 倍程度である。これはロード/ストア回路、加算器、乗算器の部分の制御はほぼ 1 倍と考えられるが、主記憶へのアクセスがクロック数で比較すると S-820 の方が多いことによる。



要素並列型演算パイプライン方式ありの場合は、2つに分けられる。1つはベクトル処理の分割起動を行わない場合（S-810と同じ起動方式）で、図7.2では「要素並列あり2」で示してある。もう1つはS-820で導入したベクトル処理の分割起動を行う場合で、図7.2では「要素並列あり1」で示した。整数予測や状態方程式のループのように、ループ内の演算回数やメモリアクセスの多いものについてベクトル処理の分割起動方式は、特に大きな効果があることがわかる。

S-820ではS-810と比較してロード/ストア回路、加算器、乗算器の数をそれぞれ2倍にしてあるので方式性能比は2倍前後が目安となる。「要素並列あり1」（ベクトル処理の分割起動方式あり）の場合で比較すると、いずれのループもほぼ2倍になっている。2倍を越えているループ（差分、差分予測、内積2）は、いずれも要素並列型演算パイプライン方式によって、ロード/ストア回路、加算器、乗算器の利用度をS-810より2倍以上に高めているループである。

## 7.2 総和型演算の性能評価

総和型演算の高速化例として、表 7.2 にリバモア・24 ループのなかの内積計算を用いたループの高速化例を示す。

表 7.2 後処理演算器で高速化を図った内積演算の性能例(リバモア・24 ループ)

ループ内容	S-820	S-810
kernel 3 内積演算 DO 3 K=1,1001 3 Q=Q+Z(K) x X(K)	838.7 MLOPS (3.91) [2.23]	214.5 MLOPS (1.0)
kernel 4 内積演算 DO 4 J=5,1001,5 TEMP=TEMP - XZ(LW) x Y(J) 4 LW=LW+1	258.5 MLOPS (2.98) [1.70]	86.7 MLOPS (1.0)

( ) 内は絶対性能比、[ ] 内は方式性能比

S-820 は S-810 に比べマシン・サイクルで 1.75 倍向上している。従って、絶対性能比をマシン・サイクル比 1.75 で割ったものを方式性能比として表わした。方式性能比が 2 倍近いのは、要素並列パイプライン方式と後処理演算器で高速化を図った結果である。

### 7.3 巡回型演算の性能評価

表 7.3 に専用制御演算器で高速化を図った巡回型演算の性能例を示す。表 7.2 と同様、S-820 は S-810 に比べマシン・サイクルで 1.75 倍向上しているのに、絶対性能比をマシン・サイクル比 1.75 で割ったものを方式性能比をして表わした。方式性能比が 2 倍以上なのは、専用制御演算器で高速化を図った結果である。

表 7.3 専用制御演算器で高速化を図った演算の性能例(リバモア・14 ループ)

ループ内容	S-820	S-810
kernel 5 【三角化消去 下三角】一次巡回演算 DO 5 I=2,998,3 X(I) = Z(I) x (Y(I) - X(I-1)) X(I+1) = Z(I+1) x (Y(I+1) - X(I)) 5 X(I+2) = Z(I+2) x (Y(I+2) - X(I+1))	114.6MLOPS (4.85) [2.77]	23.6MLOPS (1.0)
kernel 6 【三角化消去 下三角】一次巡回演算 DO 6 I=3,999,3 I=1000-J+3 X(I) = X(I) - Z(I) x X(I+1)) X(I-1) = X(I-1) - Z(I-1) x X(I)) 6 X(I-2) = X(I-2) - Z(I-2) x X(I-1))	111.8MLOPS (4.26) [2.43]	26.2MLOPS (1.0)
kernel 11 【総和】逐次加算 DO 11 K=2,100 11 X(K)=X(K-1) + Y(K)	98.4MLOPS (5.85) [3.34]	16.8 MLOPS (1.0)

( ) 内は絶対性能比、[ ] 内は方式性能比

また 5.3.2 で示したように、kernel 5 と kernel 6 は今回新しく開発したコンパイラ技術により初めてベクトル化が可能になった。従来のコンパイラ技術ではこれらはスカラ処理であった。スカラ処理の性能は、

	S-820	S-810
kernel 5	18 MFLOPS	5 MFLOPS
kernel 6	12 MFLOPS	5 MFLOPS

であるので、この 2 つの kernel は新しいコンパイラ技術の助けにより大きな性能向上を図ることが出来た。

#### 7.4 主記憶スループットの評価

図 7.2 と図 7.3 に主記憶スループットの評価を示す。これはベクトル・ロード命令およびベクトル・ストア命令をそれぞれ一つのロード・パイプライン(M0)あるいはロード/ストア・パイプライン(M1)を用いて実行したものをシミュレーションにより評価したものである。

横軸はベクトル・ストライドで、ベクトル要素が飛び飛びの場合を変化させて評価した。縦軸は主記憶スループット (%) であるが、一つのパイプライン(M0 あるいは M1)のピーク・スループットは 8 GB/秒であるのでこの値を 100% とした表示にしてある。

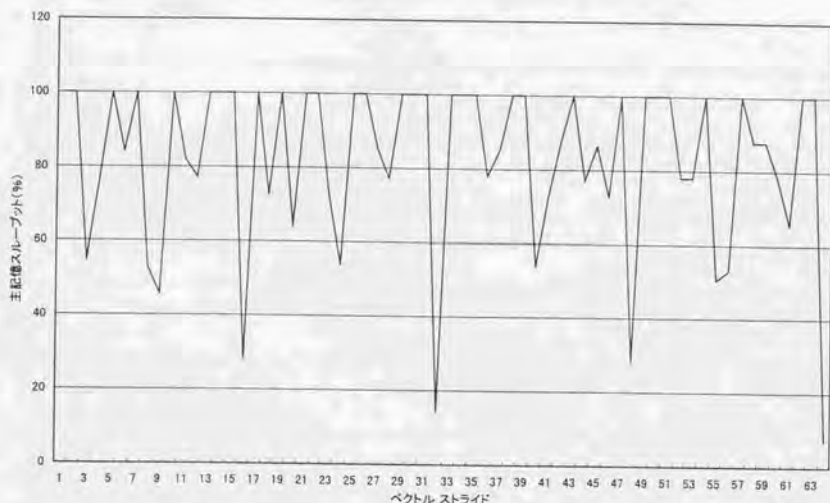


図 7.2 ベクトル・ロード命令の主記憶スループット

図 7.2 はベクトル・ロード命令の主記憶スループットを示している。プログラムによく現れるベクトル・ストライドが 1 および 2 の場合は主記憶スループット 100% である。これは 6.2.3 にて記述した「バンク・グループ番号モディファイ方式」がうまく効いていることを示している。

ベクトル・ストライドが3の場合、主記憶スループットが約半分にまで低下する。これはベクトルの8要素が並列に連続してアクセスされるため、一つのセクション番号が同一バンク・グループに集中するための競合による低下である。

またベクトル・ストライドが、16, 32, 48, 64 の場合は主記憶スループットの低下が著しい。これはベクトルの8要素が並列に連続して同一バンクにアクセスされるためである。S-820では主記憶に高速SRAMを用いているが、やはり同一バンクのアクセスによるスループット低下はやむを得ない。

図 7.3 にベクトル・ストア命令の主記憶スループットを示す。この図において、ベクトル・ストライドが1および2の場合は主記憶スループット 100%であるが、ベクトル・ストライドが4以上の場合は、良くても主記憶スループットが60%になってしまう。

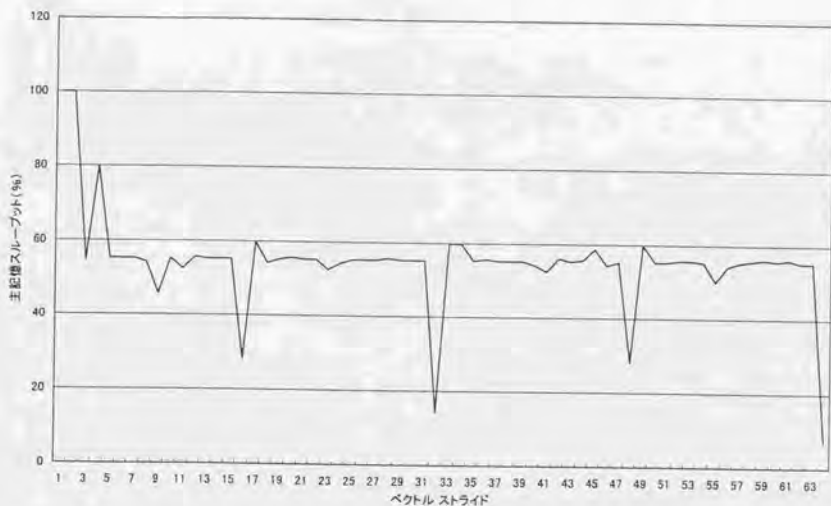


図 7.3 ベクトル・ストア命令の主記憶スループット

これは、ベクトル・ストアの動作に関して、主記憶に書き込むアドレスとスカラ・プロセッサが有しているキャッシュのアドレスを比較するための回路がボトル・ネックになるためである。ベクトル・ストアは主記憶に直接データを書き込むため、他のキャッシュと内容の一致を保証するためのアドレス比較回路が必要になる。



このアドレス比較回路をさらに多重に持つことにより、回路のボトル・ネックを避けることは可能であるが、プログラムを分析した結果、ベクトル・ストアの動作に関してはベクトル・ストライドが1および2の場合が多いため、上記のような制御とした。



## 7.5 総合性能評価

以上述べた技術による S-820 の性能改善のようすを表 7.4 に示す。性能は先にも述べたリバモアループで評価している。表中の番号が 14 ループのループ番号を示している（一部のループに対する性能は割愛してある）。数値の単位は MFLOPS (Million Floating-point Operations Per Second、 $10^6$  演算/秒) である。

表 7.4 S-820 の性能評価 (リバモアループ)

番 号	ループ名	S-820			S-810	SX-2	備 考
		要素並列あり		要素並列 なし			
		分割起動 あり	分割起動 なし				
1	流体	856	818	372	239	754	
2	内積 1	812	772	406	261	415	
3	内積 2	854	812	333	214	562	演算数 2
4	帯型連立一次	225	225	104	67	124	
5	三角化消去	114	114	114	23	14	巡回型演算
6	三角化消去	111	111	111	26	15	巡回型演算
7	状態方程式	883	797	407	262	804	ベクトル長 120
8	P.D.E. 積分	214	188	174	112	153	ベクトル長 20
9	整数予測	802	662	362	232	719	ベクトル長 100
10	差分予測	220	218	86	55	130	
11	総和	98	98	98	16	24	巡回型演算
12	差分	408	368	142	91	254	演算数 1
13	2次元粒子	17	17	13	5	8	スカラ演算中心
14	1次元粒子	24	24	20	9	24	スカラ演算中心
平 均 (性能比)		402 (3.4)	373 (3.2)	191 (1.6)	115 (1)	286 (2.5)	SX-2 286 CRAY-2 69

単位：MFLOPS ( $10^6$  演算/秒)

SX-2：情報処理 Vol.28 No.11 1987 による

14 ループの性能の単純平均をとると、S-820 の性能は 384MFLOPS と S-810 の 112MFLOPS に対し 3.4 倍向上しており、実効性能は 3 倍以上改善している。この性能は要素並列型演算パイプライン方式、分割起動方式を採用した場合のものである。

いずれも採用しない場合の性能は、175MFLOPS と 1.6 倍の向上にとどまる。この性能向上はほぼマシンサイクルの向上によるものである。要素並列型演算パイプライン方式のみを採用した場合は、355MFLOPS と採用しない場合に比べて 2 倍、S-810 に比べて 3.2 倍の性能向上となる。

要素並列型演算パイプライン方式による 2 倍の性能向上は、表 7.1 に示した S-820/S-810 の演算器の利用率の向上の 2 倍とちょうど合う。さらに分割起動方式を加えると、なしの場合に比べて 7% の性能向上となる。これは一見性能向上度が低いように見えるが、この理由は 14 ループのベクトル要素数が既して多い（ループあたりの平均要素数が 386）ためである。要素数が少ない場合の分析は後に述べる。

巡回型演算の高速化はこの論文の大きな目的の 1 つである。ループ 5, 6, 11 がこの演算に属するが、新しいコンパイラ技術の開発により、高速ベクトル化が実現できた。他社と比較してもこの点が大きな優位技術の 1 つとなった。

一方、他社と比較すると、この当時世界最高といわれる日電の SX-2 の 283MFLOPS と比較して 1.4 倍の性能となっている。この値は必ずしも十分ではないが、今後のコンパイラ等の改善により性能をさらに向上させることが可能である（最終的な実測値は 422.7MFLOPS となり、1.47 倍となった<sup>7.3)</sup>）。Cray Research 社の CRAY-2 の性能は 69MFLOPS と予想外に低い性能である。これは CRAY-2 が 4 台の CPU から構成され、全体で S-820 と同等の性能である。しかも主記憶容量 2GB を実現するためにアクセス時間の遅い DRAM を採用（他の CRAY システムは Bipolar RAM もしくは SRAM を使用している）、これが、性能のボトルネックになっていると思われる。

次に表 7.4 の各ループを更に詳細に分析し、S-810 の課題がいかに解決されたかについて調べる。

#### (1) 項数が少ない演算時の性能

ループ 3（内積）、ループ 12（差分）は演算数が 2 または 1 で、S-820 では 1 つの演算器しか稼動しない（S-810/S-820 の乗算器にはこれに専用の加算器が付加されており、内積演算は乗算器のみで演算できる）ループである。これらのループの性能は、要素並列型演算パイプライン方式によって、

ループ 3	333MFLOPS	→	812MFLOPS	(2.4 倍)
ループ 12	142MFLOPS	→	368MFLOPS	(2.6 倍)

と、平均性能で2倍以上に向上している。

### (2) 短ベクトル演算時の性能

ループ 7(状態方程式)、ループ 8(P.D.E.積分)、ループ 9(予測)はベクトル長(処理ベクトル要素数)がそれぞれ120、20、100と少なく、短ベクトル演算の好例である。

分割起動方式によってこれらのループの性能は、

ループ 7	797MFLOPS	→	883MFLOPS	(1.14 倍)
ループ 8	188MFLOPS	→	214MFLOPS	(1.14 倍)
ループ 9	662MFLOPS	→	802MFLOPS	(1.21 倍)

と11~21%向上している。

### (3) スカラ演算性能

ベクトル化ができない部分は、スカラ性能が大きな役割を果たす。リバモア・ループの中では、ループ 13(2次元粒子推進)とループ 14(1次元粒子推進)はループの中の一部にベクトル演算が不可能な部分があり、コンパイラはこの部分を分割(ループ分割)してスカラ演算で実行させている。実行時間の内訳を見ると、90%以上がこのスカラ処理に費やされており、このループの性能と比較してもほぼスカラ性能の向上度合いを知ることができる。すなわち、

ループ 13	5MFLOPS	→	13MFLOPS	(2.6 倍)
ループ 14	9MFLOPS	→	20MFLOPS	(2.2 倍)

で、2倍以上のスカラ性能向上が実現されているといえる。

## 第 8 章 今後の高速処理方式に関する考察

### 8.1 はじめに

スーパーコンピュータに代表される高性能コンピュータの性能はここ 20 年の間に著しく向上した。1976 年にセイモア・クレイが設計した記念すべき CRAY-1(160MFLOPS)が出荷され、その後のスーパーコンピュータの「さきがけ」としての大きな役割を果たした。

この 20 年の間歩みは、まずベクトル型のスーパーコンの発展があり、次に RISC 型のマイクロプロセッサの登場により WS が高速化され、それに伴って並列プロセッサも実用化される様になった。

その結果、20 年後の昨年 1996 年末には 1TFLOPS を超える性能が報告されるまでになった。この様な急速な性能の向上はいくつかの技術開発によっているが、大きくは

- ①半導体技術
- ②アーキテクチャ
- ③並列化技術 (ハード・ソフト含む)

などであろう。

ここでは今日までの技術の進歩を振り返ると共に、今後の技術動向について触れてみる。

### 8.2 半導体技術の進歩と今後の動向

半導体技術は DRAM に代表されるメモリ集積度の技術進歩とマイクロプロセッサに代表される高速論理技術の進歩がある。図 8.1 に DRAM の集積度の進歩を示す。20 年前は 16Kb の DRAM が出発めばかりであり、今日の 16Mb の最盛期と比較すると 1000 倍以上の進歩である。DRAM の集積度は 3 年で 4 倍のペースが続いている。

またマイクロプロセッサの集積度も同じく図 8.2 に示すように、Moore の法則と呼ばれる 1 チップ内のトランジスタ数が 18 ヶ月毎に 2 倍になる法則が続いている。現在の集積度は数百万トランジスタであるが、21 世紀には 1 億トランジスタを超えると予測される。

このような半導体の進歩はいつまで続くのであろうか？図 8.1 に半導体のデバイス技術の最小加工寸法(Feature Size)の推移を合わせて示す。現在は  $0.35\mu\text{m}$  から  $0.25\mu\text{m}$  の時代であるが、今後は  $0.1\mu\text{m}$  近くまで 21 世紀に入ってもこの進歩は続くと考えられる。 $0.1\mu\text{m}$  を超えた際の半導体技術の限界は

- ①トランジスタの大きさの限界
- ②配線幅の減少に伴う配線抵抗の増大 (高速化の壁)
- ③チップの発熱の増大に伴う冷却



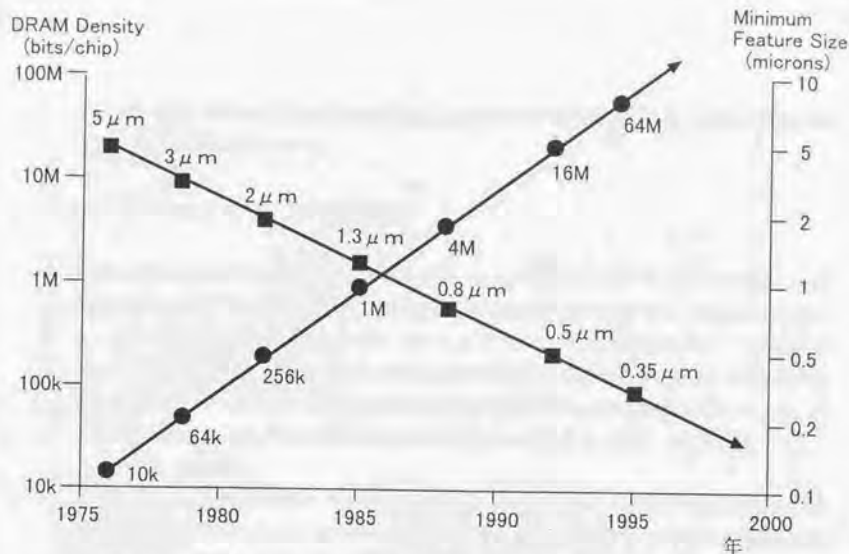


図8.1 半導体の進歩とDRAMの集積度の進歩

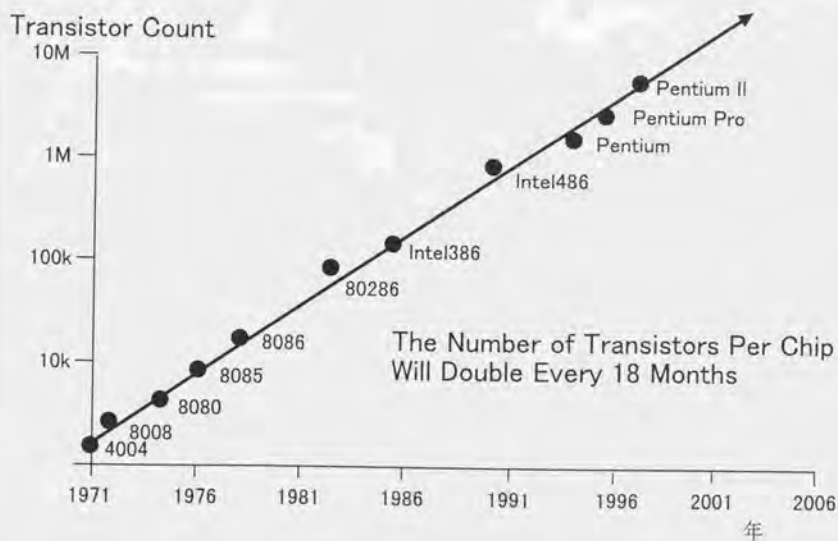


図8.2 Mooreの法則

などであるが、今後 21 世紀に向けて精力的な研究および技術開発によりこれらの課題が解決される事が期待されている。

### 8.3 アーキテクチャの動向と今後の課題

'70 年代の後半から '80 年代の後半にかけてベクトル型のスーパーコンピュータは著しい進歩を遂げた。図 8.3 に連立一次方程式(LINPACK)1000 元のプログラムにおけるベクトル・プロセッサ(単体かつ Cray の例)とマイクロプロセッサの性能を示す。この時代はマイクロプロセッサは CISC の時代で性能において大きな差があった。しかし '80 年代の後半から RISC マイクロプロセッサが大きな発展を遂げてきた。この結果 '90 年代に入り、ベクトル型のスーパーコンとの性能差も著しく接近する状況になった。

図 8.4 にベクトル型のスーパーコンと RISC マイクロプロセッサを用いたコンピュータの構造の概略を示す。ベクトル型のスーパーコンはベクトルレジスタと呼ばれる 100KB 程度の高速のレジスタを持ち、演算器と主記憶との間にあって「ベクトル・ロード」と呼ぶ高速データ転送を、演算器がデータを使用する前に予め行うことにより性能を上げるのが特徴である。これに対し、多くの RISC マイクロプロセッサを用いたコンピュータは数 MB の容量のキャッシュを持っている。キャッシュと演算器との間は高速のデータ転送を行い性能を上げているが、キャッシュと主記憶との間のデータ転送能力は比較的低い。

従って、図 8.5 に示すように、RISC マイクロプロセッサを用いたコンピュータは、データがキャッシュに入っている時は高い性能が得られるが、データがキャッシュから溢れると性能が著しく低下する傾向にある。

最近の RISC マイクロプロセッサは非常に高速になっており、大容量のキャッシュを持つことにより高い性能が得られるが、大規模な科学技術計算では、大容量のキャッシュを持っても、データがキャッシュから溢れるためこの問題を解決する必要がある。このための技術としては、主記憶からキャッシュへの「プリフェッチ技術」および主記憶からレジスタあるいはキャッシュへの「プリロード技術」がある。

「プリフェッチ技術」はハードウェアによるものと、ソフトウェアによるものが考えられる。実際の RISC マイクロプロセッサでの評価も始まったばかりであるが<sup>\*)</sup>、効果も確認され始めており今後の重要な技術課題である。

これに対し、「プリロード技術」は、日立の SR2201 並列プロセッサシステムにおける RISC マイクロプロセッサの中に「疑似ベクトル機能」を持つアーキテクチャ<sup>\*)</sup>として実

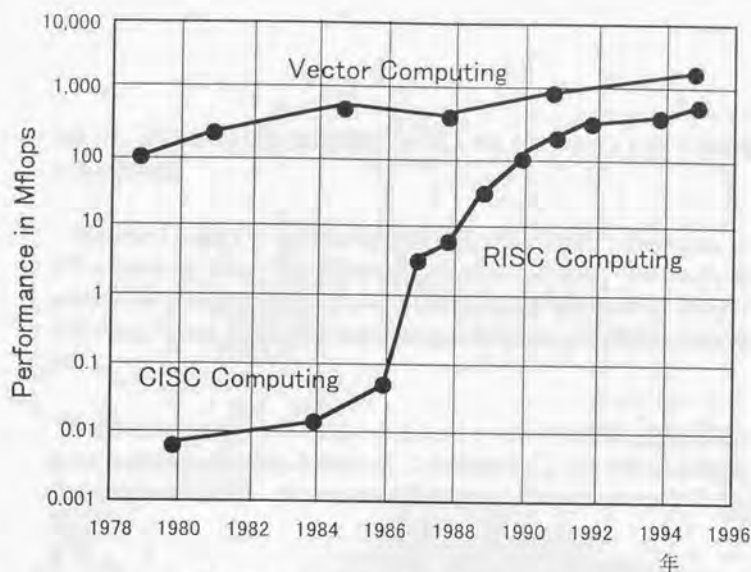


図8.3 Linpack 1000 x 1000 性能

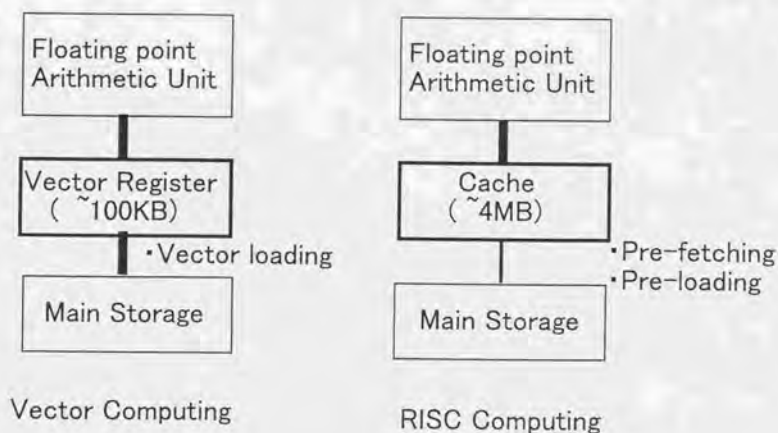


図8.4 ベクトル処理とRISC処理

用化され、ベクトル処理的な処理を行うことにより図 8.5 に示すように、この問題の解決を図っている。

「疑似ベクトル機能」や「ベクトル処理的な処理」という記述をしているのは、アーキテクチャの点から言えば、ベクトル命令を持っていないからである。そのかわり、浮動小数点レジスタを 128 個に拡張し、このレジスタに対する「プリロード命令」および「ポストストア命令」を持っており、演算命令と組み合わせて、ベクトル処理的と同様の処理をしている。

今後の RISC アーキテクチャには、このような「ベクトル処理的な処理」を行うアーキテクチャが組み込まれて行くと思われる。この意味からもここ数十年にわたるベクトル演算処理方式の研究開発は、その成果が次の世代に新しい形で引き継がれて行くことであろう。

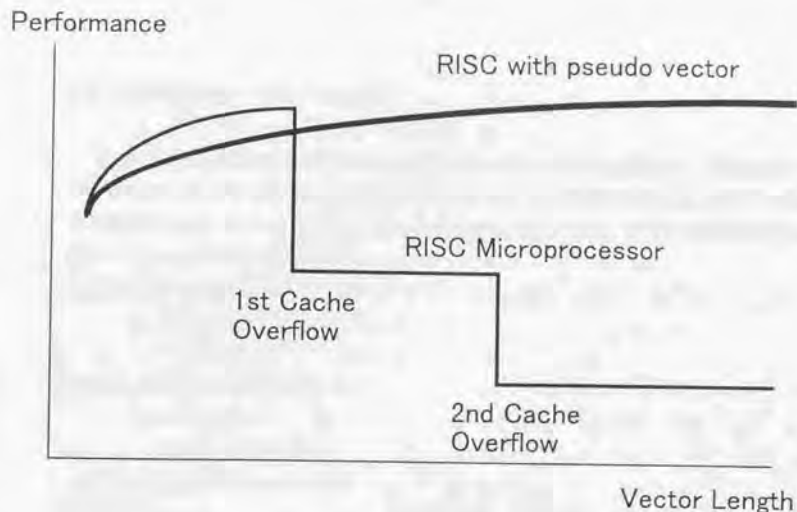


図8.5 RISC性能とベクトル処理性能

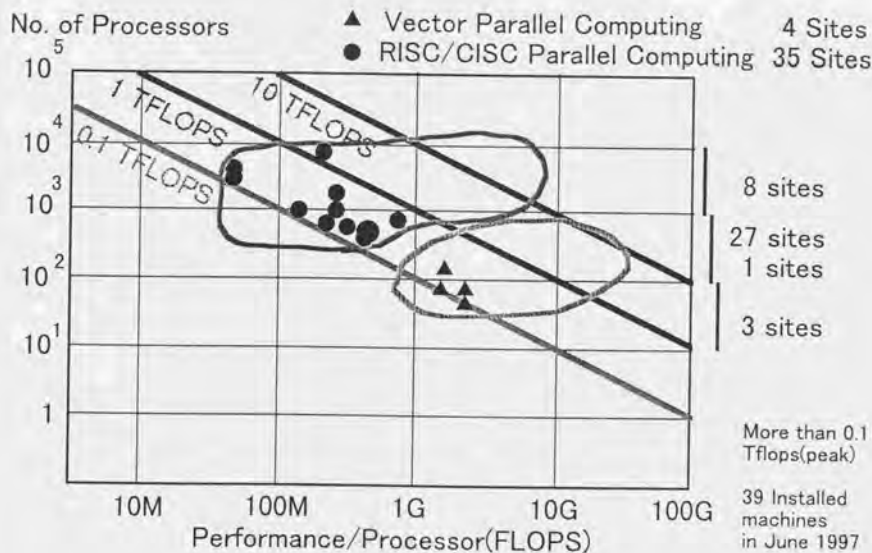


図8.6 TFLOPSへのアプローチ



#### 8.4 今後の高性能コンピュータの動向

以上、この分野における技術動向と市場規模を考察してきた。図 8.6 に TFOLPS へのアプローチと称して 1997 年 6 月現在における全世界での 100GFLOPS 以上のコンピュータの分類をした。39 セットのコンピュータが設置されているが、すべて主記憶分散型の並列コンピュータである。

RISC 型またはベクトル型で分類すると、

- ・ RISC 型 35 セット
- ・ ベクトル型 4 セット

である。また並列台数で分類すると、

- ・ 100 台未満 3 セット
- ・ 100 台～1,000 台未満 28 セット
- ・ 1000 台～10,000 台未満 8 セット

となっている。

従って、今後の高性能コンピュータは大きな技術進歩が予測される DRAM と RISC 型マイクロプロセッサ技術を中心に発展して行くであろう。図 8.7 に現在の高性能コンピュータの現状、問題点および今後の課題を纏めた。

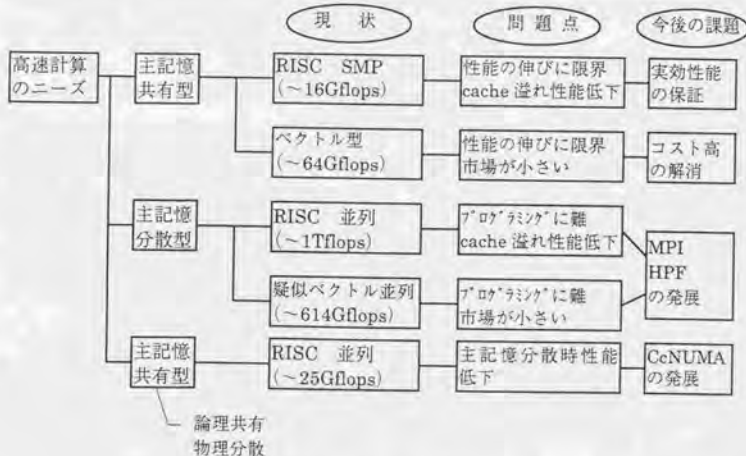


図 8.7 高性能コンピュータの課題

現状の高性能コンピュータは、大きく5つのタイプに分類される。

- ① RISCの主記憶共有SMP(Symmetric Multiprocessor)
  - ② ベクトル型的主記憶共有SMP(Symmetric Multiprocessor)
  - ③ RISCの主記憶分散並列
  - ④ ベクトル型的主記憶分散並列
  - ⑤ RISCの主記憶物理分散論理共有並列
- ①、②は主記憶共有型である。③、④は主記憶分散型並列、⑤は主記憶は物理的には分散しているが論理的には共有型の並列である。

主記憶共有型はここ数十年の高性能コンピュータを引っ張ってきた方式であり、ハード・ソフト共に技術的にも成熟の域に達している。しかし性能の伸びに限界があり(主記憶を共有するための記憶制御のハードに限界がある)近未来では~100GFLOPSが到達可能な性能であろう。

しかし、高性能コンピュータに対する近未来の要求性能は数~数十TFLOPSの領域に来ており、主記憶共有型では実現が難しく、主記憶分散型並列方式に解が求められている。主記憶分散型並列方式の最大の問題点は、プログラミングが難しいという点であろう。このためにメッセージ・パッシングのプロトコルを高位レベルで規定したMPIが普及し始めている。図8.6に100GFLOPS以上の39セットのコンピュータが設置されていることを示したが、この多くのサイトではMPIが使用されている。

HPFはメッセージ・パッシングのプロトコルを意識しないで良いという点では優れているが、規格の決定が遅れたことおよびコンパイラの性能がまだ十分でない点があり、MPIほどは普及していない。しかし主記憶分散型並列方式の普及の為にはこのような高位言語が必須であり今後普及することを願っている。

また将来は、自動並列化によるコンパイラの開発や、ハードによる実効性能の高い主記憶の物理分散・論理共有技術の開発など技術的ブレイクスルーに期待したい。

これからの高性能コンピュータの開発は、これらの技術動向を取り入れて、大規模な科学技術計算を効果的に解くアーキテクチャの開発とソフトを含む並列処理技術の開発などが中心になって進むであろう。

今後の技術開発に期待したい。

## 第9章 結 言

本論文では、科学技術計算分野におけるコンピュータの高速化技術であるベクトル演算処理方式を採用したベクトル・プロセッサにおいて、シングル・プロセッサの高速化に関し、主としてアーキテクチャやプロセッサの論理方式上の研究を対象とした。

まず科学技術計算におけるベクトル演算処理方式の効果について論じた。ベクトル演算処理アーキテクチャとしては、

- ・主記憶上のデータを直接演算器に入力して演算する方式
- ・主記憶上のデータをベクトル・レジスタで受けて演算する方式

の2つが考えられる。これらを比較し、主記憶と演算器との間のデータ転送量の少ない、現在の「ベクトル・レジスタ演算方式」が使い勝手よりもむしろ性能重視の点で採用された理由を説明した。

次にベクトル演算処理方式が有効に働くための、ベクトル演算の種類とベクトル化率およびそれを実現するアーキテクチャであるベクトル命令の種類と命令数について論じた。アーキテクチャに関しては、ベクトル化率の点から見ると、

- |                           |          |
|---------------------------|----------|
| ・第1世代の ASC, STAR-100 などでは | 約 50%    |
| ・第2世代の Cray-1, IAP などでは   | 約 60~70% |
| ・第3世代の S-810 などでは         | 約 80%    |

となり、技術としてはほぼ成熟の域に達してきたと考えられる。

第3章では、プロセッサの実現方式であるベクトル演算処理方式の論理方式について述べた。まず、ベクトル演算処理方式の概念と高速処理の原理について述べ、第3世代の S-810 において実現したベクトル処理演算方式とこの分析を行い、課題と後継機種である第4世代の S-820 の開発技術を明らかにした。

第4章では、まず S-820 において実現したベクトル演算処理方式について述べ、さらに S-820 の特長技術の一つである、要素並列型演算パイプライン方式について詳細を述べた。これにより、ベクトル・プロセッサにおいて、シングル・プロセッサの高速化技術としては、要素並列型演算パイプライン方式が、従来技術の方式と比較して方式比で約2倍の性能向上があり、極めて有効であることを示した。

さらに第5章においては、要素並列型演算パイプライン方式の高速性を享受できない、要素間に相互依存性のある処理の重要性和高速化方式について述べ、特に S-820 にて開発

し実現した内積、総和型演算処理と巡回型演算処理についてその高速化技術と効果について述べた。

第6章において、短いベクトル要素の処理性能を向上させるための技術である、ベクトル処理の分割起動方式についてその高速化技術と効果について述べた。また演算制御と共に、主記憶とのデータの制御方式も重要である。ベクトル処理の主記憶制御方式について述べた。これも S-820 において開発し実現した技術である。

第7章では、本研究で開発し、実現した各種の高速処理方式の性能評価を行なった。特に、要素並列型演算パイプライン方式の評価、および要素間に相互依存性のある処理として内積、総和型演算と巡回型演算の性能評価について詳細を述べた。また主記憶スループットの評価と共に、総合的な性能評価としてリバモア・14 カーネルを中心とした分析を行い、効果を確認した。

本研究の主たる成果は、1987年12月に出荷された HITAC S-820 システムとして具体化され、実用に供された。本スーパーコンピュータは、東京大学、北海道大学をはじめとする大規模計算センタや、分子科学研究所、高エネルギー物理学研究所、気象研究所をはじめとする国立研究所、自動車、機械、化学、電気などの製造メーカ、および気象庁などにおける大規模計算ユーザに計算パワーを提供し、本研究の効果が、多くの実社会のシステムでも確認された。

今後も一層高性能の高速コンピュータの開発が期待されている。第8章では、今後の高速処理方式についての考察を行った。シングル・プロセッサの高速化方式技術としては、本研究で開発した「要素並列型演算パイプライン方式」において、技術としてはほぼ成熟の域に達してきたと考えられるが、今後の高速処理方式のキーワードは「プロセッサ間の並列処理」いわゆる「並列コンピュータ」であろう。実際、全世界の 100GFLOPS 以上のコンピュータはすべて「並列コンピュータ」であり、今後もこの方向であろう。

では現在の「並列コンピュータ」は、技術的な観点から本研究の「ベクトル・プロセッサ」の第何世代に相当するのであろうか？ 筆者の主観で言わせていただければ、第1世代の後半かあるいは第2世代の前半であろうか。

「ベクトル・プロセッサ」の第1世代はベクトル演算処理の技術的効果は認めながらも、使い易さと価格の点で大きくは普及しなかった。ところが第2世代になって、半導体の進歩（高速半導体 RAM の登場）と相俟って「ベクトル・レジスタ演算方式」が本格的に導入され、コンパイラの「自動ベクトル化コンパイラ」の開発と共に、「ベクトル・プロセッサ」は大きく普及し、今日に至った。



この点から考えると、「並列コンピュータ」が大きく普及するための技術的ブレイクスルーは

- ・自動並列化コンパイラ技術
  - ・ハードによる主記憶の物理分散・論理共有技術
- などユーザから見た飛躍的な使い勝手の改善であろうか。

半導体技術は、DARM やマイクロプロセッサを中心に今後 10 年間は現在のベースでの技術開発が進むと考えられる。「並列コンピュータ」技術も今後 10 年以内にこれらの課題が解決され、社会に大きく普及して行くことを願ってやまない。



## 謝 辞

本論文は、1974年より研究開発に着手した、ベクトル演算処理アーキテクチャおよびプロセッサの実現方式である論理方式に関して、1987年12月に出荷したベクトル・プロセッサ HITAC S-820 システムの研究開発の成果を、東京大学 工学系 研究科 情報工学専攻 田中 英彦教授のご指導のもとにまとめたものである。

本論文の作成に当たっては、田中教授から、ご多忙中の中、格段のご指導・ご鞭撻をいただいた。さらに、情報工学専攻 井上 博充教授、情報工学専攻 武市 正人教授、電子工学専攻 近山 隆教授、電子情報工学専攻 喜連川 優教授の方々からは、極めて有益なご助言をいただいた。

筆者は、ベクトル演算処理アーキテクチャおよびプロセッサの実現方式である論理方式の研究開発および設計を、(株)日立製作所において1974年1月より1996年8月までの22年余に亘って推進してきた。推進に当たっては、小高 俊彦 取締役 汎用コンピュータ事業部長、堀越 彌 (株)日立情報システムズ専務取締役、中澤 喜三郎 電気通信大学 教授、古舘 賢一 常務取締役、ほか多くの方々からご指導・ご助言をいただいた。

1974年1月より内蔵ベクトル演算方式の研究開発に着手したが、推進は上記の方々に加え、中田 育男博士 (現 図書館情報大学 教授)、梅谷 征雄博士 (現 静岡大学 教授)、高 貴 隆司氏、面田 耕一郎氏らをはじめとする、中央研究所、神奈川工場、ソフトウェア工場の多数の方々との共同研究開発設計の成果である。

さらに1979年8月より本格的なベクトル演算方式であるスーパーコンピュータの研究開発に着手したが、最初の製品化である1983年10月に出荷したベクトル・プロセッサ HITAC S-810 アレイプロセッサシステムの推進は上記の方々に加え、長島 重夫博士、安村 通晃博士 (現 慶応大学 教授)、鳥居 俊一氏、稲上 泰弘氏、田中 義一氏の中央研究所の方々、松浦 嗣夫氏、村山 浩氏、後藤 二三男氏、阿部 仁氏、畠山 靖彦氏、桐生 芳雄氏、磯部 章氏、らの神奈川工場の方々、青山 智夫 (現 宮崎大学 教授)、磯部 敏子氏らの (株)日立コンピュータエンジニアリングの方々、およびソフトウェア工場のオペレーティングシステム、コンパイラ開発担当の多数の方々との共同研究開発設計の成果である。

本論文は、主としてS-810の後継機種である1987年12月に出荷したベクトル・プロセッサ HITAC S-820 システムの研究開発の成果であるが、これの推進は上記の方々に加え、中央研究所の中川 貴之氏、玉置 由子氏ら、神奈川工場の和田 英夫氏、石井 幸一氏、青

木 雄二氏、中川 八穂子氏、柏山 正守氏、深川 正一氏、古川 正男氏、矢澤 茂子氏らの多数の方々との共同研究開発設計の成果である。

本研究内容を論文にまとめることに関しては、小高 俊彦 取締役 汎用コンピュータ事業部長、中村 道治 中央研究所 所長、長島 重夫 同副所長、をはじめ、社内各位のご支援・ご激励をいただいた。また、筑波大学との CP-PACS 開発中には、日立製作所の OB でもある当時の中澤 喜三郎 教授および中田 育男教授からもぜひ論文にまとめるようご指導をいただいた。

以上のように名前を挙げた人々に加え、本研究内容の製品化に関しては、非常に多くの方々の参加と努力の結集である。これらのすべての方々へ心より感謝の意を表する次第である。

## 参考文献

### 第1章

- 1-1) Purcell, C. J., "The Control Data STAR-100 Performance Measurement," Proc. Of NCC, Vol. 43, pp. 385-387, May 1974.
- 1-2) Watson, W. J. and Carr, H. M., "Operational Experiences with TI Advanced Scientific Computer," Proc. Of NCC, Vol. 43, pp. 389-397, May 1974.
- 1-3) Russell, R. M., "The Cray-1 Computer System," Communications of ACM, Vol. 21, No. 1, pp. 63-72, Jan. 1978.
- 1-4) Lincoln, N. R., "Technology and Design Tradeoffs in the Creation of a Modern Supercomputer", IEEE Trans. On Computers, Vol. C-31, No. 5, pp. 349-362, May 1982.
- 1-5) 三輪 修、乾 範男、久米 宣明、内田 啓一郎、「FACOM 230-75 アレイプロセッサ」、情報処理、Vol. 29, No. 1, pp. 410-415, 1977 年 4 月。
- 1-6) 小高 俊彦、柳田 友厚、高貫 隆司、梅谷 征雄、河辺 峻、堀越 彌、「HITAC M-180 内蔵アレイプロセッサ」、日立評論、Vol. 60, pp. 451-456, 1978 年 6 月。
- 1-7) 河辺 峻、梅谷 征雄、高貫 隆司、村山 浩、小高 俊彦、「HITAC M-200H 内蔵アレイプロセッサ」、電子通信学会技術研究報告 (電子計算機研究会)、EC80-79, 1981 年 3 月。
- 1-8) Takanuki, R., Umetani, Y. and Nakata, I., "Some Compiling Algorithm for an Array Processor", 3rd USA-Japan Computer Conference, pp. 273-279, 1978.
- 1-9) Fernbach, S., 「米国でのスーパーコンピュータの利用分野—現状と将来」日経コンピュータ、1981.12.28, pp. 116-125.
- 1-10) 小高 俊彦、小林 二三幸、河辺 峻、長島 重夫、「最大性能が 630MFLOPS で 1G バイトの拡張記憶が付くスーパーコンピュータ HITAC S-810」、日経エレクトロニクス、1983.4.11, pp. 159-184.
- 1-11) 小高 俊彦、長島 重夫、河辺 峻、村山 浩、後藤 二三男、「スーパーコンピュータ HITAC S-810 アレイプロセッサシステム」、日立評論、Vol. 65, pp. 541-546, 1983 年 8 月。
- 1-12) 平栗 俊男、田畑 晃、槌本 隆光、田口 尚三、「マシン・サイクル 7.5ns を達成した並列パイプライン方式のスーパーコンピュータ FACOM VP」、日経エレクトロニクス、1983.4.11, pp. 131-155.
- 1-13) 田村 宏、内田 啓一郎、岡本 哲郎、「スーパーコンピュータ FACOM VP のハードウェア」、FUJITSU, Vol. 35, No. 4, pp. 465-475, 1984 年 4 月。
- 1-14) 古勝 紀誠、渡辺 貞、近藤 良三、「最大性能 1.3GFLOPS、マシン・サイクル 6ns のスーパーコンピュータ SX システム」、日経エレクトロニクス、1984.11.19,

pp. 237-272.

- 1-15) 河辺 峻、小林 二三幸、村山 浩、桐生 芳雄、半田 洋光、田上 文一、後藤 志津雄、青山 智夫、「シングル・プロセッサで最大性能 2GFLOPS の S-820」、日経エレクトロニクス、1987.12.28, pp.111-125.
- 1-16) 「FUJITSU VP2000 シリーズ特集号」、FUJITSU, Vol. 41, No. 1, 1990 年 1 月.
- 1-17) 「22 GFLOPS のスーパーコンピュータを日電が発売」、日経エレクトロニクス、1989.5.1, pp.74-75.
- 1-18) 「4 プロセッサ構成で 32GFLOPS のスーパーコンピュータを日立が発売」、日経エレクトロニクス、1992.4.13, pp.88-89.
- 1-19) Thompson, J. R., "The CRAY-1, the CRAY X-MP, the CRAY-2 and Beyond: The Supercomputers of the Cray Research", SUPERCOMPUTERS Class V Systems, Hardware and Software, Sidney Fernbach (editor), North-Holland, pp. 113-136, 1986.  
邦訳: 長島 重夫訳: 「スーパーコンピュータ」、パーソナルメディア、1988 年 4 月.
- 1-20) 寺内 和也、「主記憶 2 G バイトで液浸冷却方式の CRAY-2 スーパーコンピュータ」、日経エレクトロニクス、1985.12.16, pp.195-209.
- 1-21) 「最大性能 0.5~4GFLOPS の CRAY Y-MP シリーズを発売」、日経エレクトロニクス、1989.4.3, pp.86-87.
- 1-22) Vacca, T., Resnick, D., Frankel, D., Bach, R., Kreilich, J. and Carison, D., 「液体窒素冷却の CMOS LSI をスーパーコンピュータに使う」、日経エレクトロニクス、1987.9.7, pp.153-161.

## 第 2 章

- 2-1) 小高 俊彦、河辺 峻、「超高速演算の動向」、情報処理、Vol. 21, pp. 927-937, 1980 年 9 月.
- 2-2) 河辺 峻、小高 俊彦、「日立製作所の計算機開発の歴史と展望—科学技術計算の高速化アーキテクチャについて—」、情報処理学会 計算機アーキテクチャ研究会、1984 年 3 月.
- 2-3) 浦城 恒雄、小高 俊彦、河辺 峻、「国産コンピュータはこうして作られた—日立製作所—」、bit 別冊、1985 年 9 月.
- 2-4) 河辺 峻、後 保範、長島 重夫、小高 俊彦、「大規模計算とベクトル計算機」、情報処理学会第 26 回全国大会予稿集 6N-1, pp. 165-166, 1983 年 3 月.
- 2-5) 小高 俊彦、小林 二三幸、河辺 峻、長島 重夫、「最大性能が 630MFLOPS で 1G バイトの拡張記憶が付くスーパーコンピュータ HITAC S-810」、日経エレクトロニ



クス、1983.4.11, pp. 159-184.

- 2-6) 長島 重夫、小高 俊彦、「スーパーコンピュータの展望」、日立評論、Vol. 69, No. 12, pp.1-6, 1987年12月.
- 2-7) Umetani, Y., et al, 「An Analysis on the Vector Operation to Scientific Programs and the Determination of an Effective Instruction Repertoire」, 3rd UJCC, pp. 270-276 (Feb. 1980).
- 2-8) 長島 重夫、田中 義一、「スーパーコンピュータ」、オーム社、コンピュータアーキテクチャシリーズ、1992年11月.

### 第3章

- 3-1) 小高 俊彦、河辺 峻、「HITAC S-810 アレイプロセッサシステム」、HITAC ユーザ、1982年11月.
- 3-2) 小高 俊彦、小林 二三幸、河辺 峻、長島 重夫、「最大性能が630MFLOPSで1GBバイトの拡張記憶が付くスーパーコンピュータ HITAC S-810」、日経エレクトロニクス、1983.4.11, pp. 159-184.
- 3-3) 小高 俊彦、長島 重夫、河辺 峻、村山 浩、後藤 二三男、「スーパーコンピュータ HITAC S-810 アレイプロセッサシステム」、日立評論、Vol. 65, pp.541-546, 1983年8月.
- 3-4) 長島 重夫、河辺 峻、「日立スーパーコンピュータ S-810 - HITAC S-810 における並列パイプライン演算方式 -」、Computer Today, pp. 29-30, 1984年7月.
- 3-5) Nagashima, S., Inagami, Y., Odaka, T. and Kawabe, S., 「Design Consideration for a High-speed Vector Processor : The HITACHI S-810」、Proceeding of IEEE International Conference on Computer Design : VLSI in Computers ICCD'84, pp. 238-242, (Oct. 1984).
- 3-6) 河辺 峻、「HITAC S-810」、JECC ジャーナル、1986年11月.
- 3-7) 小高 俊彦、河辺 峻、「HITAC S-810/S-820」、コンピュータロール、No. 20, pp. 96-101, 1987年10月.
- 3-8) Horikoshi, H., Nagashima, S., and Furumaya, K., 「SUPERCOMPUTER: HITACHI S-810」、JARECT, Vol. 18, Computer Science and Technologies, T. Kitagawa(editor), OHMSHA, LTD. and North-Holland, pp. 57-69 (1988).



## 第4章

- 4-1) 河辺 峻、村山 浩、松浦 嗣夫、桐生 芳雄、青山 智夫、稲上 泰弘、「スーパーコンピュータ HITAC S-820 システム」、日立評論、Vol. 69, pp.1103-1108, 1987 年 12 月。
- 4-2) 河辺 峻、小林 二三幸、村山 浩、桐生 芳雄、半田 洋光、田上 文一、後藤 志津雄、青山 智夫、「シングル・プロセッサで最大性能 2GFLOPS の S-820」、日経エレクトロニクス、1987.12.28, pp.111-125.
- 4-3) Odaka, T., Kawabe, S. and Wada, H., 「Development of Hitachi supercomputer S-820 system」, Third International Conference on Supercomputing, Boston, pp. 71-77, 1988.
- 4-4) Wada, H., Kawabe, S. and Odaka, T., 「Hitachi supercomputer S-820 Overview」, Proceeding of Supercomputing Europe'89, pp. 139-147, 1989.
- 4-5) Kawabe, S., Hirai, M. and Goto, S., 「Hitachi S-820 Supercomputer System」, High Performance Computing Research and Practice in Japan (Edited by Raul Mendez), John Wiley & Sons, 1992.
- 4-5) Kawabe, S., 「Hitachi S-820 Supercomputer System」, Supercomputers and Their Performance in Computational Fluid Dynamics (Edited by Kozo Fujii), Vieweg, Braunschweig, 1993.
- 4-6) 田中 義一、岩沢 京子、「ベクトル計算機のためのコンパイル技術」、情報処理、Vol.31, No.6, pp.736-743 (June 1990)
- 4-7) Umetani, Y. and Yasumura, M., 「A Vectorization Algorithms for Control Statements」、J of Information Processing, Vol. 7, No.3, pp.170-174 (1984)

## 第5章

- 5-1) 河辺 峻、小林 二三幸、村山 浩、桐生 芳雄、半田 洋光、田上 文一、後藤 志津雄、青山 智夫、「シングル・プロセッサで最大性能 2GFLOPS の S-820」、日経エレクトロニクス、1987.12.28, pp.111-125.
- 5-2) Wada, H., Ishii, K., Fukagawa, M., Murayama, H. and Kawabe, S., 「High-speed Processing Schemes for Summation Type and Iteration Type vector Instruction on Hitachi Supercomputer S-820 System」, ACM International Conference on Supercomputing, Saint-Malo, pp. 197-206, 1988.
- 5-3) D. Heller, 「Some Aspects of the Cyclic Reduction Algorithm for Block Tridiagonal Linear Systems」, SIAM J. Numr. Anal., Vol.13, No.14, pp.484-496, 1976.
- 5-4) Tanaka, Y., Iwasawa, K., Umetani, U., and Gotou, S., 「Compiling Techniques for

- First-Order Linear Recurrences on a Vector Computer」, The Journal of Supercomputing, 4, pp.66-82, 1990
- 5-5) 田中 義一、前島 英雄、「命令レベル並列計算機のためのリカレンス演算の高速化手法とコンパイラへの実装」、情報処理学会論文誌、第37巻、第9号、pp1657-1665,1996

## 第6章

- 6-1) 河辺 峻、小林 二三幸、村山 浩、棚生 芳雄、半田 洋光、田上 文一、後藤 志津雄、青山 智夫、「シングル・プロセッサで最大性能 2GFLOPS の S-820」、日経エレクトロニクス、1987.12.28, pp.111-125.
- 6-2) Wada, H., Ishii, K., Yazawa, S. and Kawabe, S., 「High-speed Vector Instruction Execution Schemes of Hitachi Supercomputer S-820 System」, Proceeding of International Conference on Parallel Processing, pp. 291-298, 1988.
- 6-3) Wada, H., Isobe, T., Furukawa, M. and Kawabe, S., 「High-speed Storage Control Schemes of Hitachi Supercomputer S-820 System」, ACM International Conference on Supercomputing, Crete, Greece, pp. 341-350, 1989.

## 第7章

- 7-1) Wada, H., Kawabe, S. and Odaka, T., 「Hitachi supercomputer S-820 Overview」, Proceeding of Supercomputing Europe'89, pp. 139-147, 1989.
- 7-2) Wada, H., Isobe, T., Furukawa, M. and Kawabe, S., 「High-speed Storage Control Schemes of Hitachi Supercomputer S-820 System」, ACM International Conference on Supercomputing, Crete, Greece, pp. 341-350, 1989.
- 7-3) Tanaka, Y., Iwasawa, K., Umetani, U., and Gotou, S., 「Compiling Techniques for First-Order Linear Recurrences on a Vector Computer」, The Journal of Supercomputing, 4, pp.66-82, 1990

## 第8章

- 8-1) 河辺 峻、「スーパーコン、汎用大型」、日経マイクロデバイス、1989年8月。
- 8-2) 河辺 峻、「スーパーコンピュータ テラフロップスを目指して」、電子情報通信学会、1989年9月。
- 8-3) 河辺 峻、「最新の半導体技術による高性能・小型化の追求と並列処理技術」、コンビ

ユータ・シミュレーション、1996年12月.

- 8-4) Nakazawa, K., Nakamura, H., Imori, H. and Kawabe, S., 「Pseudo Vector Processor based on Register-Windowed Superscalar Pipeline」, IEEE Computer Society Proceeding of Supercomputing '92, pp.641-651, 1992.
- 8-5) VanderWiel S. P., Lilja D. J., 「When Caches Aren't Enough: Data Prefetching Techniques」, IEEE Computer Society COMPUTER, pp.23-30, July 1997.
- 8-6) 河辺 峻、「次世代計算機に何を望むか - 計算機メーカーの立場から -」日本機械学会 第74期通常総会講演会資料集(V)、pp.258-259, 1997年3月.

## 執筆論文一覧

- 1) 小高 俊彦、柳田 友厚、高貫 隆司、梅谷 征雄、河辺 峻、堀越 彌、「HITAC M-180 内蔵アレイブロッセッサ」、日立評論、Vol. 60, pp.451-456, 1978年6月。
- 2) 小高 俊彦、河辺 峻、「超高速演算の動向」、情報処理、Vol. 21, pp.927-937, 1980年9月。
- 3) 河辺 峻、梅谷 征雄、高貫 隆司、村山 浩、小高 俊彦、「HITAC M-200H 内蔵アレイブロッセッサ」、電子通信学会技術研究報告(電子計算機研究会)、EC80-79, 1981年3月。
- 4) 小高 俊彦、河辺 峻、「HITAC S-810 アレイブロッセッサシステム」、HITAC ユーザ、1982年11月。
- 5) 河辺 峻、後 保範、長島 重夫、小高 俊彦、「大規模計算とベクトル計算機」、情報処理学会第26回全国大会予稿集 6N-1, pp.165-166, 1983年3月。
- 6) 小高 俊彦、小林 二三幸、河辺 峻、長島 重夫、「最大性能が630MFLOPSで1Gバイトの拡張記憶が付くスーパーコンピュータ HITAC S-810」、日経エレクトロニクス、1983.4.11, pp.159-184。
- 7) 小高 俊彦、長島 重夫、河辺 峻、村山 浩、後藤 二三男、「スーパーコンピュータ HITAC S-810 アレイブロッセッサシステム」、日立評論、Vol. 65, pp.541-546, 1983年8月。
- 8) 河辺 峻、小高 俊彦、「日立製作所の計算機開発の歴史と展望—科学技術計算の高速化アーキテクチャについて—」、情報処理学会 計算機アーキテクチャ研究会、1984年3月。
- 9) 長島 重夫、河辺 峻、「日立スーパーコンピュータ S-810 — HITAC S-810 における並列パイプライン演算方式 —」、Computer Today, pp. 29-30, 1984年7月。
- 10) Nagashima, S., Inagami, Y., Odaka, T. and Kawabe, S., 「Design Consideration for a High-speed Vector Processor: The HITACHI S-810」、Proceeding of IEEE International Conference on Computer Design: VLSI in Computers ICCD'84, pp.238-242, (Oct. 1984)。
- 11) 浦城 恒雄、小高 俊彦、河辺 峻、「国産コンピュータはこうして作られた—日立製作所—」、bit 別冊、1985年9月。
- 12) 河辺 峻、「HITAC S-810」、JECC ジャーナル、1986年11月。
- 13) Nagashima, S., Nakagawa, T., Omota, K., Miyamoto, S., Kawabe, S. and Tsuchiya, Y., 「Hardware Implementation of VELVET on the HITACHI S-810 Supercomputer」、Proceeding of IEEE International Conference on Computer-Aided Design ICCAD'86, pp.390-393, (Nov. 1986)。



- 14) 鳥居 俊一, 小島 啓二, 吉住 誠一, 河辺 峻, 高橋 政美, 久代 康雄, 「リレーショナル・データベースの処理速度向上を図る CPU 内蔵型データベース・プロセッサ」, 日経エレクトロニクス, 1987.2.9, pp.185-206.
- 15) Torii, S., Kojima, K., Yoshizumi, S., Sakata, A., Takamoto, Y., Kawabe, S., Takahashi, M., Ishizuka, T., 「A Relational Database System Architecture Based on a Vector Processing Method」, Proceedings of the Third International Conference on Data Engineering, pp.182-189, (Feb. 1987).
- 16) 小高 俊彦, 河辺 峻, 「HITAC S-810/S-820」, コンピュータール, No. 20, pp.96-101, 1987年10月.
- 17) 河辺 峻, 村山 浩, 松浦 嗣夫, 桐生 芳雄, 青山 智夫, 稲上 泰弘, 「スーパーコンピュータ HITAC S-820 システム」, 日立評論, Vol. 69, pp.1103-1108, 1987年12月.
- 18) 河辺 峻, 小林 二三幸, 村山 浩, 桐生 芳雄, 半田 洋光, 田上 文一, 後藤 志津雄, 青山 智夫, 「シングル・プロセッサで最大性能 2GFLOPS の S-820」, 日経エレクトロニクス, 1987.12.28, pp.111-125.
- 19) Odaka, T., Kawabe, S. and Wada, H., 「Development of Hitachi supercomputer S-820 system」, Third International Conference on Supercomputing, Boston, pp.71-77, 1988.
- 20) Wada, H., Ishii, K., Fukagawa, M., Murayama, H. and Kawabe, S., 「High-speed Processing Schemes for Summation Type and Iteration Type vector Instruction on Hitachi Supercomputer S-820 System」, ACM International Conference on Supercomputing, Saint-Malo, pp.197-206, 1988.
- 21) Wada, H., Ishii, K., Yazawa, S. and Kawabe, S., 「High-speed Vector Instruction Execution Schemes of Hitachi Supercomputer S-820 System」, Proceeding of International Conference on Parallel Processing, pp.291-298, 1988.
- 22) Takamine, Y., Miyamoto, S., Nagashima, S., Miyoshi, M. and Kawabe, S., 「Clock Event Suppression Algorithm of VELVET and its Application to S-820 Development」, ACM/IEEE 26th Design Automation Conference Proceedings, pp.716-719, (Jun. 1988).
- 23) 阿部 仁, 和田 英夫, 石井 幸一, 河辺 峻, 「スーパーコンピュータにおける記憶階層について」, 情報処理学会 計算機アーキテクチャ研究会, 1988年10月.
- 24) Wada, H., Kawabe, S. and Odaka, T., 「Hitachi supercomputer S-820 Overview」, Proceeding of Supercomputing Europe'89, pp. 139-147, 1989.
- 25) Wada, H., Isobe, T., Furukawa, M. and Kawabe, S., 「High-speed Storage Control Schemes of Hitachi Supercomputer S-820 System」, ACM International Conference on Supercomputing, Crete, Greece, pp. 341-350, 1989.
- 26) Torii, S., Kojima, K., Yoshizumi, S., Sakata, A., Takamoto, Y., Kawabe, S., Takahashi, M., Ishizuka, T., 「A Relational Database System Architecture Based on



- a Vector Processing Method」, Information Sciences, Vol.48, No.2, pp.135-155, Elsevier, (July 1989).
- 27) 河辺 峻、「スーパーコン、汎用大型」、日経マイクロデバイス、1989年8月。
  - 28) 河辺 峻、「スーパーコンピュータ—テラフロップスを目指して—」、電子情報通信学会、1989年9月。
  - 29) 河辺 峻、「Technology Today スーパーコンピュータとは」、電気計算、1989年11月。
  - 30) 矢島 章夫、栗原 恒弥、安生 健一、河辺 峻、青山 明夫、「スーパーコンピュータとサイエンティフィック・ビジュアルゼーション」、日立評論、1990年3月。
  - 31) 長島 重夫、中川 貴之、面田 耕一郎、宮本 俊介、河辺 峻、三善 正之、「ベクトル処理による高速論理シミュレーションの実現」、電子情報通信学会論文誌 D-1、Vol.J47-D-1, No.6, pp.360-368, 1991年6月。
  - 32) 長島 重夫、稲上 泰弘、阿部 仁、河辺 峻、「動的チェイニングによるベクトルプロセッサの実効性能の向上」、電子情報通信学会論文誌 D-1、Vol.J47-D-1, No.12, pp.836-845, 1991年12月。
  - 33) Kawabe, S., Hirai, M. and Goto, S., 「Hitachi S-820 Supercomputer System」, High Performance Computing Research and Practice in Japan (Edited by Raul Mendez), John Wiley & Sons, 1992.
  - 34) 河辺 峻、阿部 仁、「日立製作所 S-3000 シリーズ」、システム制御情報学会誌、1992年8月。
  - 35) Nakazawa, K., Nakamura, H., Imori, H. and Kawabe, S., 「Pseudo Vector Processor based on Register-Windowed Superscalar Pipeline」, IEEE Computer Society Proceeding of Supercomputing '92, 1992.
  - 36) 河辺 峻、「スーパーコンピュータ」、数学セミナ、1993年3月。
  - 37) 河辺 峻、村山 浩、「スーパーコンピュータ Hitachi S-3000 シリーズ」、計算工学、1993年6月。
  - 38) Kawabe, S., 「Hitachi S-820 Supercomputer System」, Supercomputers and Their Performance in Computational Fluid Dynamics(Edited by Kozo Fujii), Vieweg, Braunschweig, 1993.
  - 39) 河辺 峻、「最新の半導体技術による高性能・小型化の追求と並列処理技術」、コンピュータ・シミュレーション、1996年12月。
  - 40) 河辺 峻、「次世代計算機に何を望むか—計算機メーカの立場から—」日本機械学会 第74期通常総会講演会資料集 (V)、pp.258-259, 1997年3月。

