講　座

並列処理概論（3）*

村　岡　洋　一**

7. ハードウェア

並列計算機は大きく分類すると命令の並列処理とデータの並列処理の2種がある．

（1） 命令の並列処理．

命令デコーダ（演算器）が複数個あり，同時に複数の命令（演算）が並列に処理される．

（2） データの並列処理．

命令デコーダは1個であるが，1命令で複数のデータ対の演算が指示されこれが並列に処理される．

次に各々の例を示す．

7.1 命令の並列処理

7.1.1 複数演算器システム

CDC 6600/7600, IBM 360/91 システム等のように1CPU 内に複数の演算器を持つもの．通常この型の計算器は並列計算機の範ちゅうに入れないが，ここでは広義の並列計算機としてとりあげた．**図-26** がその概念である[1]．例えば文

$$E=A+B+C*D+E+F-G/H$$

の処理を考える．図のように演算器が複数あれば，

第1ステップ　$T_1=A+B$

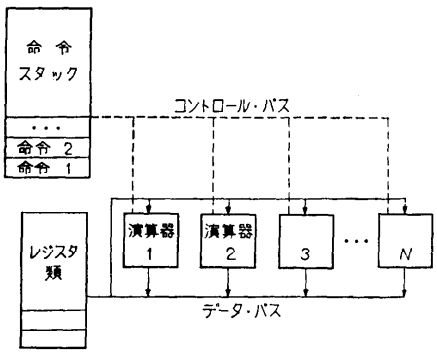


図-26　複数演算器システム

$T_2=C*D$

$T_3=E+F$

$T_4=G/H$,

第2ステップ　$T_5=T_1+T_2$

$T_6=T_3-T_4$,

第3ステップ　$T_7=E=T_5+T_6$,

のように3ステップで処理できる．複数の演算器を並列に使用するためには命令デコーダが1命令ごとに演算終了を待ってはならない．そのため図-26に示すように複数の命令を保持する命令スタックがあり，CPU 内で処理中の命令群の実行状態を管理している．

7.1.2 複数 CPU システム

マルチプロセッサ・システムである．図-20（前号 p. 124 参照）がその概念図である．複数 CPU が同一データを矛盾無く更新できるように例えばハードウェア的には Test and Set 命令が，ソフトウェア的には LOCK マクロが用意される．同時に複数 CPU から参照・更新されてはいけないデータにはロックをつける．このロックの内容が1ならデータがある CPU によって使用中の状態にあることを示す．データを使おうとする CPU はまずこのロックの内容を調べる．0であればロックを1にしてデータ使用を行なう．終了した時点で0に戻しておく．もしロックがすでに1であったならこれが0になるまで待つ．以上の処理を行なうのが LOCK マクロである．ロックはメモリ上のデータである．もしある CPU がこのロックを読んできて内容を調べている間（**図-27**（次頁参照）の ⓐ）に他の CPU もロックを読むことを許すと，2つの CPU が同時にロックの内容を0と見なしてしまう結果も生じる．このためにはⓐの処理の間，他 CPU からのメモリのアクセスを禁ずる必要がある．これを実現したのが Test and Set（略して TS）命令である．

7.1.3 複数演算器とマルチプロセッサの比較

複数演算器システムとマルチプロセッサは，両者ともユニットの数をふやすことによって処理速度を向上させようとしている点で，互に共通するところがある．前者では単一命令ストリーム，後者では異なった

---

* Parallel Processing and Parallel Processors (3) by Yoich MURAOKA (N.T.T. Yokosuka Electrical Communication Laboratories).

** 日本電信電話公社　横須賀電気通信研究所データ通信研究部

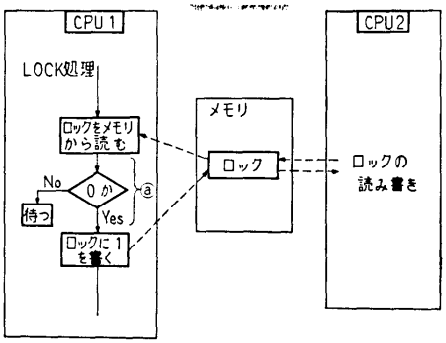


図-27 Test and Set 命令

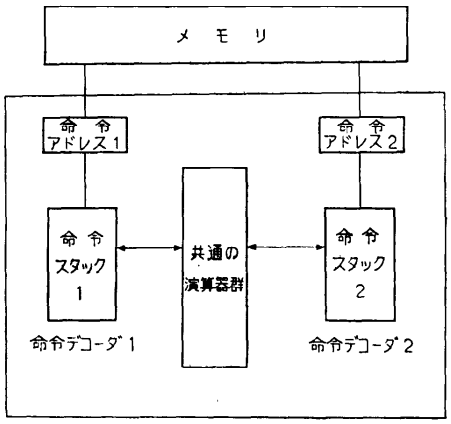


図-28 複数命令デコーダによる演算器群の共用

命令ストリームが対象となるところが主な相違である．複数演算器システムの性能を向上させるには命令の look ahead の距離（即ち CPU 内で同時処理される演算数）を大きくする必要がある．これを極端にしたものとして 1CPU に複数命令ストリームを持つシステムも提案されている[2]．このシステムは図-28 に示すように複数個（図では 2 個）の命令デコーダが共通の演算器群を共用する．従って各命令ストリームの look ahead 距離は大きくなくとも演算器は効率良く使用される．

7.2 データの並列処理

7.2.1 アレイ計算機システム

アレイ計算機の代表例としてイリアック IV を図-29 に示した[3]．イリアック IV は 1 つの命令デコーダ（CU と呼ぶ）と 64 個の演算器（PE と呼ぶ）を持つ．各演算器は自分のメモリ（PE メモリ．2k 語）を持っている．PE には図-29 のように 0 から 63 までの番号がふってあり，PE_i は $PE_{i\pm1}$ および $PE_{i\pm8}$ と直接デ

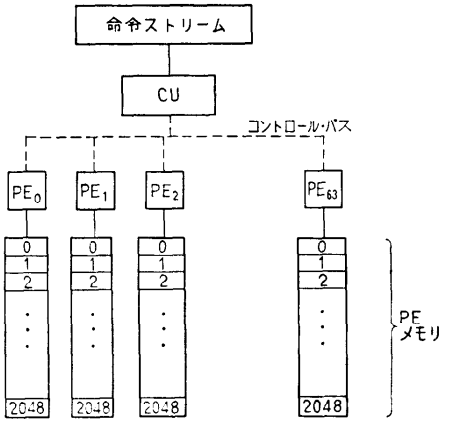


図-29 イリアック IV

ータの授受を行なうこともできる．この他に CU が共通のデータを全 PE に同時に与える（broadcast と呼ぶ）こともできる．命令は通常の計算機の命令と同等で，例えば「ADD 500」は各 PE ごとに，PE メモリの 500 番地のデータと各 PE 内のアキュムレータの内容の加算である．全 PE は CU がデコードした同一命令の演算を行なう．ただし PE ごとにモード・ビットと呼ばれる命令によって制御できるフラグがあり，これがリセットされている PE は CU からの指示を無視して演算を行なわない．番地は PE メモリごとに 0 から 2k まで割りふられている．従って「ADD 500」の命令は，各 PE が自分の PE メモリの 500 番地の内容を読みだし自分のアキュムレータに加える処理である．インデックス・レジスタが PE ごとに存在するので，各 PE が異なった番地のデータを使うことも可能である．

イリアック IV は特に行列演算等に適している．例えば 64 要素を持つベクトル A と B について，A の要素 A_i を PE_i の 0 番地へ，B の要素 B_i を PE_i の 1 番地へ各々格納すれば，1 ステップで A と B の加算が行なえる．

7.2.2 パイプライン・システム

例として浮動小数加算処理を考える．浮動小数加算処理を次の 4 つの基本処理に分解する．

（1） exponent の引き算．
（2） （1）の結果に従って fraction の位置合せ．
（3） fraction の加算．
（4） 結果の正規化．

この 4 基本処理を図-30 （次頁参照）に示すように 4 つの関数ユニットで直列に行なうとする．各関数ユニ

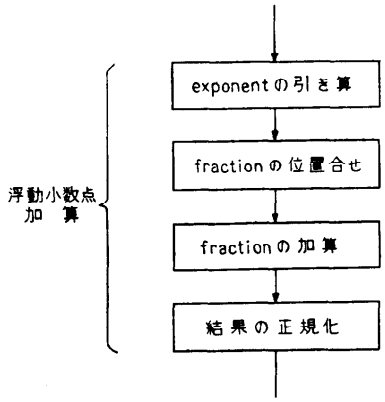


図-30　浮動小数点加算処理

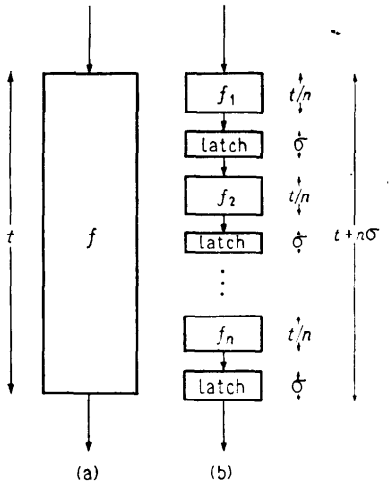


図-31　パイプラインの概念

ットの処理時間を u とする．もし各関数ユニットがlatchを持ち他ユニットと独立に処理が行なえるなら，u 秒ごとに新しいオペランド対を入力できることになる．これを一般化したのが図-31である．同図(a)は通常の演算ユニットであり処理に t 秒かかる．このユニットを n 個のサブ・ユニットに分割し，各サブ・ユニットごとにlatchをつけたのが同図(b)である．各サブ・ユニットの処理時間を t/n, latch のための時間を σ とする．2つの性能を比較すると，

(a)のユニット：

処理時間	t (秒)
スループット	$1/t$ (1/秒)

(b)のユニット：

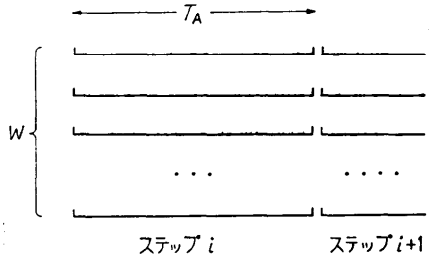
処理時間	$t+n\sigma$ (秒)
スループット	$1/(t/n+\sigma)$ (1/秒)

となる．ここでスループットは，単位時間に入力できるオペランド対の数を示す．(b)のユニットは処理時間は大きくなったがスループットがほぼ n 倍となっている．このように1つのユニットをいくつかのサブ・ユニット（ステージともいう）に分割してスループットを向上することをパイプライニングという．外部から見たところあたかも n 個の演算が並列に行なわれているように見えるので，並列計算機の一種としてとりあげた．例えば m 個の要素を持つ2つのベクトルの加算はほぼ $1/n$ の速度で行なえる．パイプライン演算器を持つシステムとしてCDC-STARがある[4]．

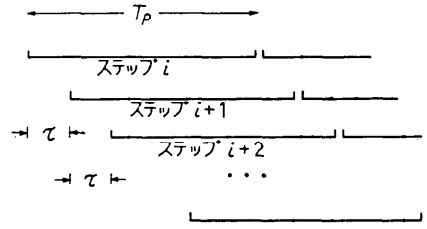
7.2.3　アレイとパイプラインの比較[4]

図-32にアレイ計算機とパイプライン計算機の動作を比較した．N 個の演算をアレイ計算機で行なうには $T_A \cdot \lceil N/W \rceil$ の時間がかかる．ここで W は演算処理ユニット(PE)数である．これに対してパイプライン計算機では $T_P+\tau(N-1)$ の時間が必要である．1番最後の演算以外は τ 秒に1個づつ処理できる．ステップ当たりの時間を2つの極端な場合について比較する．

	$N=1$	$N=\infty$
アレイ計算機	T_A 秒/ステップ	T_A/W 秒/ステップ
パイプライン計算機	T_P 秒/ステップ	τ 秒/ステップ



(a)　アレイ計算機のタイミング



(b)　パイプライン計算機のタイミング

図-32　アレイとパイプラインの比較

もし T_A/W が τ より小さくかつ T_A が T_p より小さければ，任意の N に対してアレイ計算機の処理時間の方がパイプライン計算機よりも小さくなる．

アレイ計算機のスループット（単位時間当たりの演算数）は W/T_A であり，パイプライン計算機のスループットは $1/\tau$ である．

次に両者の効率を比較する．もし $N=1$ ならばアレイ計算機の効率は $1/W$ であり，パイプライン計算機のそれは τ/T_p となる．任意の N に対してはアレイ計算機の効率は $N/(W\lceil N/W\rceil)$ であり，パイプライン計算機では $N/(T_p/\tau+N-1)$ となる．

どちらの計算機が良いかは対象とする問題のタイプにも依り一概にはいえない．ただアレイ計算機では W の値を PE の追加により原理的ではいくらでも大きくできるのに対して，パイプライン計算機の τ の値にはある限り(例えば論理素子のゲート遅延)がある．従って大きな問題に対してはアレイ計算機で対処する方が有利である．その反面小さな問題に対してはアレイ計算機では効率が悪くなる恐れがある．

7.3　その他の並列処理計算機

以上の2つのクラスの他に並列処理計算機の範ちゅうに含まれているものにアソシアティブ・メモリ・システムがある．**図-33** に1例を示した．各メモリ語にはコントロール・モジュール（アレイ計算機の PE とほぼ同等．ただし構造はより単純である）が付いている．メモリ語ごとにキィが含まれている．コントロール・ユニットは命令を解読し，全メモリ語にキィとデータを与える (broadcast する)．各メモリ語はこの broadcast されたキィと自分のキィを比較し，比較条件（例えば key equal, key high 等）が満足された場合のみ broadcast された処理を行なう．アレイ計算機との違いはこのキィ処理にある．例えばマトリックスの演算等で要素のインデックス（$A_{i,j}$ の i と j の値）をキィとして使えば，インデックスの処理やメモリ・スペースの維持等行なわなくとも良く，簡単に並列処理が実現できることになる．

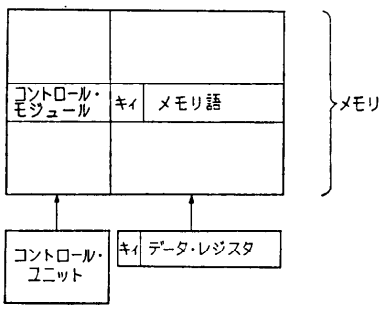


図-33　アリシアティブ・メモリ・システム

8.　並列処理の適用性

以上述べた並列処理技術をまとめて **表-1** に示した．

計算機の性能はスループットと応答時間の2つで評価できる．従来の多重処理（CPU と入出力の処理のオーバラップ）をはじめとするシステム技術は，どちらかといえばこのうちのスループット向上（すなわちシステム各装置の使用率向上）を目的としていた．これに対して並列処理は応答時間短縮を主目的とする技術である．この観点から以上に説明した並列処理技術を見なおしてみる．

便宜上，計算機の応用分野を，(i)科学技術計算，(ii)事務計算，(iii)システム・プログラムの3つに分類し，各々についてその特長を並列性の観点から比較したのが **表-2**（次頁参照）である．同表からもわかるように，科学技術計算はその性格上並列処理に適したものが多い．科学技術計算のアルゴリズム，モデル等を開発するのが物理学者や化学者であって，計算機を道具として使おうとする人がほとんどであろうから，並列処理言語の使用をこれらの人々に期待するのは難しいと思われる．従ってこの分野ではプログラム中からの並列性検出アルゴリズムを活用した並列計算機用コンパイラは歓迎されよう．これに対してシステム・プログラム中から DO 文の並列処理を発見することはそれ程期待できない．システム・プログラム中の処理の大部分は制御表等のポインタ処理であって，繰返し型の処理は少ない．従ってこの分野で並列計算機用コンパイラが活用されることは，少なくとも近い将来には考えられないであろう．タスク・レベルの並列処

表-1　並列処理技術の一覧

項目＼レベル		命令レベル	文レベル	タスクレベル
プログラマが指定	プログラミング言語	・特に実例は無い	・DO ループの並列処理 (TRANQUIL の for sim 文). ・文の並列処理 (FORK-JOIN)	・PL/I の TASK 指定.
	アルゴリズムの実例	・特に実例は無い	・偏微分方程式. ・行列演算.	・オペレーティング・システム.
従来言語プログラムを解析		・Tree height reduction アルゴリズム.	・DO 文の解析.	・特に実例は無い
計算機の実例		・IBM 360/91. ・CDC 6600.	・ILLIAC IV ・CDC STAR	・マルチプロセッサ

表-2 並列処理の適用性

分野	例	性格	プログラマ	並列性の特長	適当なシステム
科学技術計算	・行列計算 ・偏微分方程式	・繰返し処理 ・CPU処理が大きい ・種類が多い ・規模(スタティック・ステップ)は大きくない	・科学者	・DO文の並列処理	・アレイ・システム ・パイプライン・システム
事務計算	・給与計算	・文字処理 ・ファイル処理		?	?
システム・プログラム	・銀行システム ・座席予約システム	・ポインタ処理 ・I/O処理が大きい ・規模が大きい	・専門家(システム・プログラマ)	・タスク・レベル	・マルチプロセッサ

分である．並列処理はCPU処理時間の短縮に特に有効であるから，CPUリミテッドな傾向の強い科学技術計算の方が，I/Oリミテッドな傾向のシステム・プログラムよりも，より並列処理に適しているといえる．事務計算は科学技術計算とシステム・プログラムの中間に位置するといえよう．その並列処理への適用性はまだ未知である．

以上の適用範囲内で考えるならば，DO文の並列処理を含む文レベルの並列処理は並列性検出アルゴリズムとしてコンパイラが分担すべきで，タスク・レベルの並列処理はPL/I等のように利用者に指定を一任して充分であろう．使用される計算機も科学技術計算には文レベルの並列処理に適したアレイ計算機等が，システム・プログラムにはタスク・レベルの並列処理に適したマルチプロセッサが良いであろう．演算レベルの並列処理は応用分野の別を問わず活用されていくと考えられる．このレベルの並列性検出は既にいくつかのコンパイラの最適化の一部としてとり入れられている[5]．

N台の処理装置を持つ並列計算機の能力（速度で比較して）がN倍にならない主な理由として次の2つがあげられる．

(1) ソフトウェア的に．

Amdahl等が並列処理に反対した一番大きな理由は，「プログラム中には並列処理に適さない部分が多く存在し，並列処理システムを効率良く使用できない」ということであった[6]．同一の問題を直列処理した場合と並列処理した場合の処理時間を各々T_sとT_pと書く．並列計算機の処理効率eを，

$$e=T_s/(N\cdot T_p)$$

と定義する．Nは処理装置数である．今，図-34の如く直列処理を並列処理したときβ_kの割合いはk個の処理装置のみしか使えず，$(1-\beta_k)$の割合いは全ての処理装置が使えるとする．並列処理でも直列処理でも

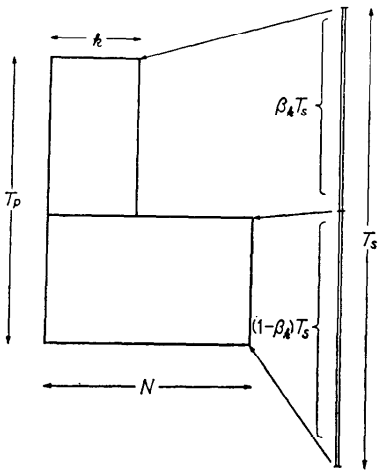


図-34 並列処理の効率

実行される全演算数は同一とすれば，

$$T_p=\beta_k(T_s/k)+(1-\beta_k)(T_s/N)$$

となり，

$$e=1/(1+\beta_k(N/k-1))$$

となる．例えば$k=1$，$N=33$，$\beta_k=1/16$とすれば$e=1/3$となる．すなわち$e=1/3$となるためには，T_sの15/16の時間は33個全ての処理装置が使われていなければならない．Amdahlはこの様な議論を使い，並列処理の有用性を否定した．しかし，6.2.3項(前号 p. 144 参照)の測定例が示すとおり，一般には$k=1$ではなく，処理ステップiにおいては$k(i)\geq 1$個の処理装置を使用できるため使用効率は上の例ほど極端には悪化しない．

(2) ハードウェア的に．

並列計算機の性能を低下させる一番大きな原因は，メモリ・アクセス競合である．通常のマルチプロセッサ(プロセッサ2，メモリ2バンクインタリーブ)において，メモリ・アクセス競合の結果平均命令実行時

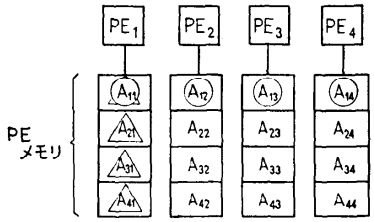


図-35 Straight mapping

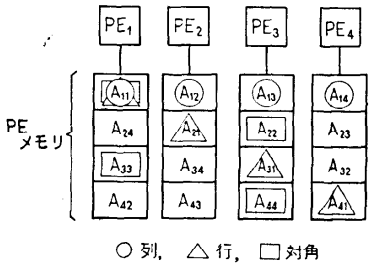


図-36 Skewed mapping

間が 10% 以上も大きくなる例も報告されている．並列計算機においてこのメモリ・アクセス競合を小さくする手段として，適切なメモリ・マッピングがある．特に科学技術計算をアレイ計算機で並列処理する場合には，行列等の構造体を対象とするのでメモリ・マッピングが容易となる．以下にイリアック IV を例に使って説明する．便宜上 PE 数を 4 として，PE メモリに 4×4 の行列 A を格納する方法を考える．一番単純な方法は**図-35** のように A の i 行を PE メモリ i に格納する方法である (straight mapping と呼ぶ[7])．この方法では同一列の要素へは全 PE が並列にアクセスできるが，同一行への要素は並列にアクセスできない．例えば第 1 行の全要素を使うには 4 回のメモリ・アクセスが必要である．この欠点を補なうために考えられたのが**図-36** の skewed mapping である[7]．これによって同一行および同一列の要素への並列アクセスは可能となるが，まだ対角要素への並列アクセスはできない．これは PE 数を 1 個ふやし 5 個にすることにより可能となる．**図-37** にこの mapping を示す[8]．この結果行，列および対角要素への並列アクセスが可能となるが，PE は常に 1 個 idle となることになる．

このように並列計算機を効率良く使用するための技術は各種開発されており，これらを活用することが必要である．

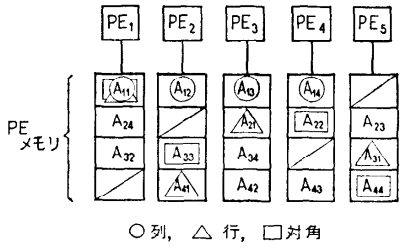


図-37 余分マッピングで

最後に並列処理による価格性能比向上の可能性にふれる．今後ますます発展する LSI 技術には repeatability の高いシステム・アーキテクチャが有利であるといわれる．これまでのアーキテクチャは高速素子をたくみに組み合わせたものであったが，今後は低速素子を多数並べて並列処理によって性能を保証するアーキテクチャも考慮すべきである[9]．例えばシステムの処理能力として，ある処理を行なうに必要なステップ数×平均命令実行時間の逆数をとり，価格性能比として処理能力/価格を考える．6.2.3 項の結果を使えば直列処理システムと並列処理システムの価格性能比 PC は各々，

$(PC)_{直列}=1/(Tg_s C_s)$,

$(PC)_{並列}=1/[(10\log_2 T)g_p(T/0.6\log_2 T)C_p]$

となる．ただし g_s と C_s は直列処理システムの平均命令実行時間と価格で，T はステップ数である．並列処理システムは，平均命令実行時間 g_p，価格 C_p の素子を $(T/0.6\log_2 T)$ 個使う．並列処理のステップ数は $10\log_2 T$ である．数値例として $T=10^6$ ステップ，$g_s=1\,\mu s$, $C_s=10^5$ ドル，$g_p=10\,\mu s$, $C_p=100$ ドルとすれば，

$(PC)_{直列}=10^{-5}$,

$(PC)_{並列}=10^{-4}$

となり，並列処理の方が直列処理システムより 1 桁程度価格性能比が良いことになる．

9. 結　言

以上に並列処理技術の概略を述べてきた．広義の並列処理技術は，ソフトウェアとしてはオペレーティング・システム，ハードウェアとしてはマルチ・プロセッサ・システムとして活用されており我々にもなじみ深いものであるが，その他の並列計算機はまだ日本で実用化された例は少なく，代表例としてパイプライン技術を応用した高速フーリエ変換処理装置を見るのみである[10]．米国では今後とも NASA 等大規模な科学

計算を必要とする組織が，新しい並列計算機の開発に力を入れていくと思われるので並列処理技術の一層の発展が期待される．

参考文献

1) R. M. Tomasulo: An Efficient Algorithm for Exploiting Multiple Arithmetic Units, IBM J., Vol. 11, No. 1, pp. 25～33 (1967).
2) M. J. Flynn, et. al: A Multiple Instruction Stream Processor with Shared Resources, in Prollel Processor Systems, Technologies, and Applications, Spartion Books, New York (1970).
3) G. H. Barnes, et. al.: The Illiac IV Computer, IEEE TC, C-17, No. 8, pp. 746～757 (1968).
4) W. R. Grahm: The Parallel & the Pipeline Computers, Datamation, pp. 68～71 (April 1970).
5) N. E. Abel, et. al.: TRANQUIL-A Language for an Array Processing Computer, SJCC, pp. 57～73 (1969).
6) G. Amdahl: Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities, SJCC, pp. 483～485 (1967).
7) D. J. Kuck: ILLIAC IV Software and Application Programming, IEEE TC, C-17, No. 8, pp. 758～769 (1968).
8) P. Budnik & D. J. Kuck: The Organization and Use of Parallel Memories, IEEE TC, C-20, No. 12, pp. 1566～1569 (1971).
9) D. J. Kuck & Y. Muraoka: Fast Computers from Slow Parts, Compcon 72 (1972).
10) H. Aiso, et. al.: A Very-High Speed Microprogrammable Pipeline Signal Processor, IFIP, pp. 60～64 (1974).

(昭和49年9月17日受付)