Vol. 27 No. 5 情報処理学会論文誌 May 1986

データフローコンピュータにおける直列処理の高速化とノイマンコンピュータとの速度比較†

曾和将容†† 羅 四維†††

本論文では，データフローコンピュータでの処理を，直列処理のみに対象をしぼり，1) データフローコンピュータにおける直列処理の高速化手法，2) この手法による高速化の効果，3) ノイマンコンピュータとデータフローコンピュータでの直列処理の速度比較を行っている．

1. まえがき

データフローコンピュータは，並列処理を自然な形で実行できることから，並列処理コンピュータとして注目をあびており，近年来活発に研究が行われている[1)~12)]．並列処理は主に，問題のプログラム記述に自由度を与えることとそれによる実行速度の向上に目的があるように思われるが，実際の問題には，並列処理と直列処理が混在しており，それゆえ，並列処理用コンピュータであるデータフローコンピュータのプログラムにも直列処理が，自然と入りこむ．また，プログラム作成において並列処理よりも直列処理が好んで用いられる場合もある．一般に，並列処理コンピュータで直列処理を行った場合，直列処理専用コンピュータであるノイマンコンピュータに比べてその処理速度が遅くなるのが普通であるので，この部分がボトルネックとなる可能性がある．したがって，データフローコンピュータにおいて直列処理部分を高速化することは重要なことである．本論文では上記のことを考慮して，以下のように研究を進める．1) データフローコンピュータにおける直列処理の高速化手法を提案する．2) 提案された手法によって，ハードウェア上どの程度の高速化が可能であるかを知るために，モデル化されたデータフローコンピュータの直列処理速度を算出し，改善度を求める．3) ノイマンコンピュータをモデル化し，ノイマンコンピュータの実行速度を表す式を求め，これをもとに2) で求められたデータフローコンピュータの実行速度と比較する．4) 具体的な数値例を得るために，ギブソン命令ミックス法を用いて命令の分布を代入し，速度比を求める．ただし，本論文では便宜上，ノイマンコンピュータという言葉を，von Neumann らが開発した当時のコンピュータに近いプリミティブなコンピュータに対して用いることとし（表1参照），これらを複数個組み合わせたマルチコンピュータシステムやアレイプロセッサ，ベクトルプロセッサなどはここに入れないことにする．

2. データフローコンピュータの直列処理時間

図1に直列処理部分を中心にしたデータフロープログラムを示す．N_1～N_n などの円はノードと呼ばれ，命令を表す．矢印はアークと呼ばれ，データの送り先を示す．一般に，データフロープログラムでは，ノード間はアークにより結ばれており，データはこのアークに沿って次のノードに送られる．そして，処理に必要なデータがそろったノード（実行可ノードと呼ばれる）にプロセッサが与えられたとき，ノードが実行される．本論文では便宜上，図1に表されているようなデータフロープログラムの直列処理部分を考察の対象とする．実際には，このような直列処理は，同じ直列処理の繰り返し，すなわちループ処理として表現されることが多いが，ここでは，簡単化と基本性能を知るために，ループは図のように直列に展開されているとする．

純粋なデータフロー処理では，データ構造体の処理プログラムにおいて，非同時性原理[14)]によって少なくともこのような直列処理部分が表れる．たとえば，図2(a)の構造体から図2(b)の構造体を作る場合，同図(c)のような並列プログラムを書くことはできない．この図で，ノード内の c や re は命令を表しており，c はコピー，re は変更を表している．また，たとえば 001 の表現は，2進木を左，左，右とたどることを表す選択子であり，この場合は要素1の位置を示

† Speedup of Serial Processing in Dataflow Computer and Its Speed Comparison with Neumann Computer by MASAHIRO SOWA (Department of Computer Science, Gunma University) and SI-WEI LOU (Department of Electronic Engineering, Northern Jiaotong University, China).

†† 群馬大学工学部情報工学科
††† 中国北方交通大学計算機科学技術系

表 1 ノイマンコンピュータとデータフローコンピュータの比較
Table 1 Comparison of Neumann computer with dataflow computer.

	ノイマン	データフロー
駆動原理	コントロール駆動	データ駆動
処理形態	直列処理専用	直並列処理
実行制御機構	基本的に暗示的 (実行順に命令を並べる．分岐命令のときは次に実行する命令を命令内で指示)	明示的 (命令内で次に実行する命令を指示)
主な構成要素	＊メモリ (プログラム，データ格納用) ＊1個のプロセッサ (プログラムカウンタを含む) ＊入出力装置	＊メモリ (プログラム格納用) ＊複数個のプロセッサ (プログラムカウンタを含まない) ＊マッチングユニット ＊入出力装置

している．同図(c)のプログラムが実行されると，Aのmのみがqに変った構造体と，Aのoのみがrに変った構造体，Aのpのみがsに変った構造体の3種類の構造体が得られるだけで，同図(b)の構造体Bとはならない．なぜならば，命令cによってAの構造体と全く同じ構造を持った，別々の構造体が3個作られ，その後いま作られたそれぞれの構造体内のただ一つのデータが，それぞれの re によって変更されるだけであるからである．正しく目的の構造体を得るには，同図(d)のように直列処理によるプログラムを書くか，構造体を分割して並列処理を行うプログラムを書かざるをえない．

データが外部から時間に依存して入力される場合や，ストリーム状のデータを取り扱う場合など直列処理にならざるをえない．また，並列プログラムでは，ディバイドアンドコンカー法がリカーシブコールによって実現されることが多いが，リカーシブコールは繰り返し計算の一種と見ることができるので，必然的に直列処理が入りこむ．また，直列処理になれたプログラマは，慣習上やプログラムの書きやすさ，見やすさなどから，ループを用いた繰り返し処理を多用する可能性が強いので，この場合にも直列処理が入りこむ．このようにデータフロープログラムでは，問題が本来持っている直列性からくる直列処理以外にも，このように非同時性原理やプログラマの好みや習慣などによる直列処理部分が表れる．しかし，それが全プログラムの何割を占め，この部分がどの程度の影響を与えるかを推察することは，データフローコンピュータが試作段階にある現時点では簡単ではない．

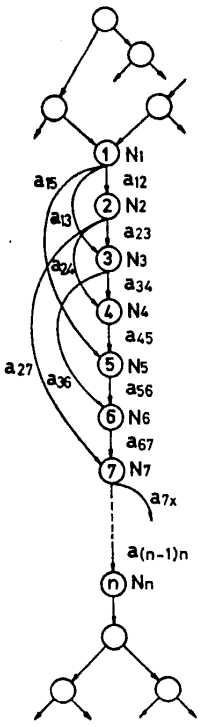


図 1 データフロープログラムの直列処理部分
Fig. 1 Serial part of data flow program.

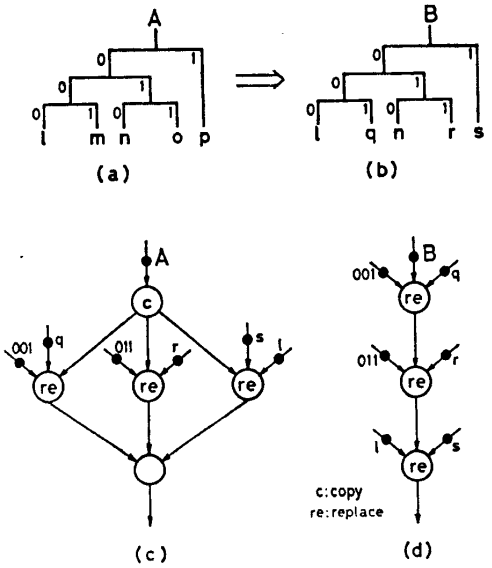


図 2 データ構造変更プログラム
Fig. 2 Data change in a data structure.

図3に本論文で対象とするデータフローコンピュータ[3]の構造を示す．NDR (node driving register)

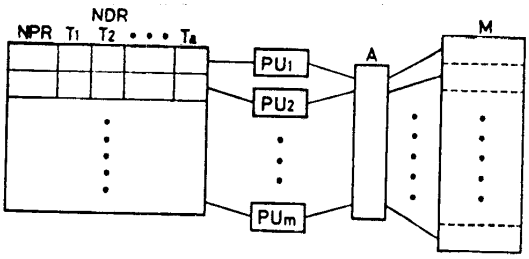


図 3 データフローコンピュータの構造
Fig. 3 Structure of data flow computer.

はトークンと呼ばれるデータを，トークンパケット TP と呼ばれるパケット形式で格納するためのレジスタ，PU_1〜PU_m はノードを実行するためのプロセッサ，Mはノードを格納するためのメモリ，Aはメモリへのアクセス競合を調停するアービタである．トークンパケット TP は，データであるトークンに，そのトークンが入力されるべきノードへのポインタ NPR (node pointer) を接続したもので，同じ NPR を持ったトークン (T_1〜T_a) は，図のように，ノードドライビングレジスタ NDR 内の1ワードにまとめられて格納される．あるノードを実行するのに十分なトークンが集められたトークンパケットは完全トークンパケット CTP (complete token packet) と呼ばれる．

ノードはノードパケットとして，命令に出力アークを接続した形でメモリに格納される．ノードの実行は次のように行われる．それぞれのプロセッサ PU はノードドライビングレジスタ NDR から完全トークンパケット CTP を取り出し (CTP fetch)，取り出した完全トークンパケット CTP のノードポインタ NPR によって示されているノードをメモリMから取り出す (NP fetch)．次に，取り出した完全トークンパケット CTP 内のトークンに対して，ノードに書かれている命令，または，処理を実行し (Execution)，処理が終ったときには，処理結果にノードに書かれているトークンの行先である出力アーク（ノードポインタ NPR）を接続し，トークンパケット TP としてノードドライビングレジスタ NDR 内に送出する (Results output)．NDR 内に送られたトークンパケット TP は，NDR の連想機能により同じノードポインタの値を持つもの同士1ワードに集められ，NDR 内に格納される．以上の動作がそれぞれのプロセッサ PU によって繰り返され，プログラムの実行が並列に進行する．

いま，簡単のために，ここで対象としているデータフローコンピュータを32ビットマシンと仮定する．

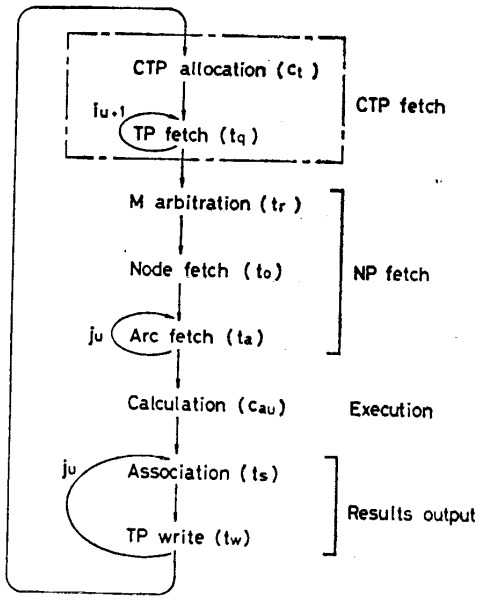


図 4 データフローコンピュータのプロセッサの実行シーケンス
Fig. 4 Operation sequence of data flow processor.

すなわち，図3の NPR，$T_1, T_2, \cdots, T_a$ がそれぞれ32ビットからなるとし，メモリMも1ワード32ビットであるとする．また，ノードパケットは命令部が16ビット，命令補助部16ビット，アーク部32ビットで表されているとする．データフロー処理では，このように仮定すると，プロセッサ動作シーケンスは図4のように書くことができる．ここでは，各ノードの入力アーク数を i_u 本，出力アークを j_u 本としている．この図で，ノードによって動作時間や回数が異なる場合には，各記号にノード名に相当する添え字 u をつけてある（例えば，u 番目のノード計算時間 C_a は C_{au}）．

n 個のノードからなる直列プログラムを実行する時間を求めるために，プロセッサの各動作に要する時間を図4の C_t, t_q などのように決める．このとき，データフローコンピュータによる n 個のノードの実行時間 t_e は

$$t_e = n(C_t + t_r + t_o) + \sum_{u=1}^{n} [t_q(i_u+1) + j_u \cdot t_a + j_u(t_a + t_w) + C_{au}] \quad (1)$$

となる．

3. データフローコンピュータにおける直列処理の高速化

一般にデータフロー処理では，ノードの実行順序がプロセッサの与えられ方と，それによって引き起こされるデータの流れによって決まるため，その順序は非

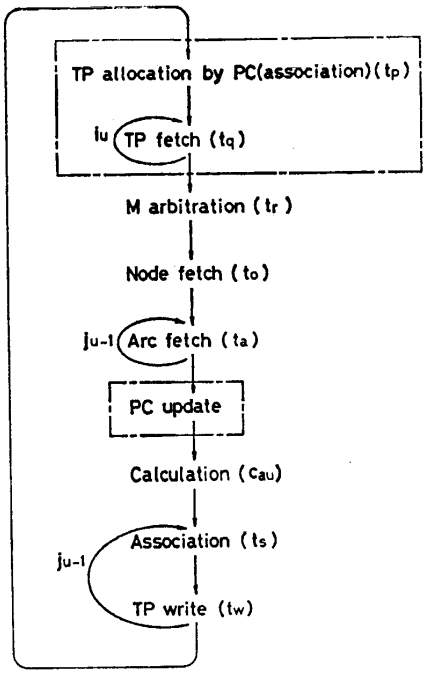


図 5 高速化されたデータフロープロセッサの実行シーケンス
Fig. 5 Operation sequence of speed-up data flow processor.

決定的であり，一意的ではない．ところが，データフロープログラムの直列処理を専用に行う場合，実行可ノードは二つ以上になることはないので，ノードの実行順序は一意的に決まってしまう．したがって，直列処理の場合に，もし，各ノードのある規則に従ってメモリに格納しておけば，たとえば図1の直列に流れるアーク $a_{12}, a_{23}, a_{34}, \cdots, a_{(n-1)n}$ を省略しても，その格納規則から次に実行すべきノードを知ることができる（性質1）．また，直列処理は，一つのノードの実行が終った時には，必ず次のノードが実行可となっているので，一連の直列プログラムに，一つのプロセッサを専用に割りつけても不都合は生じない．このことにより，一つだけではあるがノード間のデータ転送がプロセッサ内部で行える（性質2）ことになり，プログラムの実行が高速化される．すなわち，いま，ノード N_u から出るアークの数を j_u で表すことにすると，n 個のノードを実行するとき必要とされる $(j_1+j_2+\cdots+j_n)$ 回のアーク操作のうち，n 回分を省略することができる．

データフローコンピュータにおける直列処理の高速化を図るために，性質1より

1）直列処理プログラムのノードを実行順に格納する．

2）それぞれのプロセッサ内にプログラムカウンタPC を設け，ノイマンコンピュータで行われているような方法により，次に実行すべき命令を算出する．

性質2より，

3）プロセッサ PU それぞれにアキュムレータ AC を設け，隣接ノード間のデータ転送をこのアキュムレータを通して行う．

このようにすると，一つのノードを実行するためのプロセッサのシーケンスは図5のように簡略化することができる．ただし，プログラムカウンタはノードを構成する一語を取り出すごとにカウントアップされるべきであるが，ここでは一まとめにしてプログラムカウンタのアップデート (PC update) としてある．普通，プログラムカウンタのカウントアップは，メモリアクセスと並行して行われることが多いので，ここでは特に実行時間を定めていない．

この場合の n 個のノードの実行時間 t_e' は

$$t_e' = n(t_o+t_p+t_r)\sum_{u=1}^{n}[i_u\cdot t_q + t_a(j_u-1) + C_{au} + (j_u-1)(t_s+t_w)]$$
$$= t_e + n(t_p-C_t) - n(t_a+t_q+t_s+t_w) \qquad (2)$$

となる．ここでは，直列処理の高速化のためにプログラムカウンタとアキュムレータを導入したが，これらが導入されても，このコンピュータはノイマンコンピュータとは異なる．なぜならば，ノイマンコンピュータは“制御の流れ”によって命令の順序が決められる“コントロールフロー直列処理プログラム”を実行するのに対して，本データフローコンピュータでは，データの流れによって命令の実行順序が決められる“データフロー直列処理プログラム”を実行するからである．それゆえ前者では，命令中に埋めこまれたメモリ番地によりデータをフェッチする（メモリ参照）が，後者では，データが命令に流れ込むため，データのフェッチ動作はない．すなわち，本コンピュータは，プログラムカウンタとアキュムレータを持っていても原理的にデータフローコンピュータであり，ノイマンコンピュータとは区別されなければならない．

4. ノイマンコンピュータの直列処理時間

(1), (2)式をノイマンコンピュータと比べるために，ノイマンコンピュータの直列処理時間を求める．ノイマンコンピュータでは，アキュムレータやメモリに格納されたデータが，ある命令への入力データとし

て扱われ，その命令の実行結果であるデータは，アキュムレータやメモリに格納される．また，入力データへのアクセスは“参照”によって行われるため，その入力データの格納場所に再書込みが起こらない限り，格納されたデータが変化することはない．したがって，たとえば図1のノード N_1 によって，メモリのある番地に格納された結果のデータはノード N_2 と N_3, N_5 によって用いることができる．ノイマンコンピュータでは基本的な命令の入力データは2個か1個であり，また，これらの基本命令はロードストア命令 (LSI)，レジスタ-レジスタ間命令 (RRI)，レジスタとメモリ間の演算である加減算などの算術命令 (ASI)，シフトなどの命令 (SHI)，分岐などの実行制御命令に分けることができる．

いま，簡単のために，ノイマンコンピュータを32ビットマシン，すなわち，メモリの1語長，レジスタ長，命令語などが32ビットよりなるとし，32ビット長よりなるデータを取り扱うとする．そして，このコンピュータの基本動作シーケンスを**図6**のように仮定すると各命令の実行時間は

$$\begin{aligned}&\text{SHI, RRI}: t_z=t_p+t_o+C_a\\&\text{LSI}\quad\ : t_e=t_p+t_o+t_o\\&\text{ASI}\quad\ : t_v=t_p+t_o+t_o+C_a\end{aligned}\qquad(3)$$

となる．

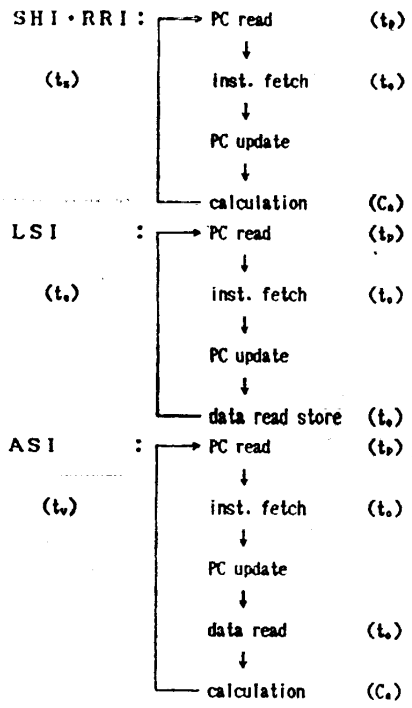


図 6　ノイマンコンピュータの実行シーケンス
Fig. 6　Operation sequence of Neumann computer.

ただし，図6では命令のデコードなど直接にデータフローコンピュータとの時間比較に関係しない時間は，それぞれのシーケンスの各ステージの一部に含まれているとし，計算時間 C_a は命令が基本命令に限られるとして，その種類によらず一定とした．また，コンピュータは基本的なもので，プリフェッチやキャッシュメモリなど高速化に対する特別な手段が用意されていないコンピュータとした．これは，ノイマンコンピュータがすでに実用化され長い研究の歴史があるコンピュータであるのに対し，データフローコンピュータはまだそれ以前にあるコンピュータであるためである．実行制御命令は図1における仮定より，ここには表れてこない．

いま，図1のプログラムを(3)式により表される命令の組み合わせによりノイマンコンピュータ用プログラムに変換したとすると，図1のプログラムを実行するのに必要な時間 t_n は

$$t_n=n'\cdot s\cdot t_z+n'\cdot l\cdot t_e+n'\cdot a\cdot t_v\qquad(4)$$

となる．

ただし，s は SHI, RRI の，l は LSI の，a は ASI 命令の出現確率であり，n' はノイマンコンピュータプログラムの命令数である．一般にノイマンコンピュータの演算に対する命令体系はデータフローコンピュータとほぼ同じとなる．ところが，ノイマン処理では直接演算の本質には関係しないロードやストアなどのメモリ操作命令が必要とされるので，この分だけ命令体系が異なる．したがって，$n<n'$ となるのが普通である．

5. 処理速度比較

データフローコンピュータとノイマンコンピュータとの処理速度比 S_n は(1)式，(4)式より

$$S_n=\frac{t_e}{t_n}=\left\{n(C_t+t_r+t_o)+\sum_{u=1}^{n}[t_q(i_u+1)+j_u\cdot t_a+j_u(t_a+t_w)+C_{au}]\right\}\Big/(n'\cdot s\cdot t_z+n'\cdot l\cdot t_e+n'\cdot a\cdot t_v)\qquad(5)$$

となる．

一方，データフローコンピュータにおける直列処理速度の改善比 S_d は(1)式，(2)式より

$$S_d=\frac{t_e}{t_e'}=\frac{t_e}{t_e+n(t_p-C_t)-n(t_a+t_q+t_s+t_w)}=\frac{1}{1+\dfrac{[t_p-(C_t+t_a+t_q+t_s+t_w)]}{t_e}}\qquad(6)$$

となる.

また，高速化されたデータフローコンピュータとノイマンコンピュータの処理速度比 S_o は

$$S_o=\frac{t_e'}{t_n}=\frac{t_e+n(t_p-C_t)-n(t_a+t_q+t_s+t_w)}{n'\cdot s\cdot t_s+n'\cdot l\cdot t_e+n'\cdot a\cdot t_v} \quad (7)$$

となる.

これらの処理速度比の具体的な値を概念的に知るために，ギブソン命令ミックス[13]を利用して，具体的な数値例を求める．ギブソン命令ミックスを利用するにあたって，本論文の主旨に合うように次のような仮定を導入する．1) ギブソン命令ミックスの命令を基本命令に限定し，2) 分岐命令とインデックス操作命令はないものとする．また，3) 浮動小数点演算をすべて固定小数点演算に換算する．1)，3) はコンピュータの基本性能を知ることからの仮定で，2)は図1における仮定から導かれる．これら三つを仮定することにより**表2**に示すような科学技術計算を行う場合の命令分布が得られる．ノイマンプログラムでは，データフロープログラムよりロード，ストアなどのメモリ参照命令分だけ命令数が多いと考えられる．ノイマンプログラムの命令数 n' はこの表より，$n'=n/0.52$ となる．ノイマンコンピュータの命令は主に2アドレス命令と1アドレス命令からなる．それゆえ，図1のプログラムで入力アークの最大数を2と仮定するとシフト命令と論理命令の NOT だけが1入力アーク命令に相当し，他は2入力命令に相当する．このことを考慮して，ギブソン命令ミックスより1入力アーク命令と2入力アーク命令分布を求めると表2のようになる．**表3**より入力アークの総数は

$$0.88\times2\times n+0.12\times1\times n=1.88n$$

となる.

表 2 命令分布
Table 2 Instruction distribution.

LSI	48%
ASI	26%
RRI SHI	26%

表 3 入力アーク数分布
Table 3 Distribution of the number of input arcs.

2入力	88%
1入力	12%

いま，式(1)から式(7)における時間のうち，単純なレジスタアクセス時間である t_p を $t_p=u$（uは単位時間）とし，レジスタや論理回路の組合せで行う操作時間 C_t, t_q, t_r, t_s, t_w をすべて等しいとする．また，メモリ操作に関する時間 t_a, t_o を等しいとし，$u=1-n$ に対し $C_{au}=C_a$ とすると，処理速度比は**図7**，**図8**のようになる．図7は計算時間 C_a をメモリアクセス時間 t_a に等しくした時，レジスタ組み合わせ回路の動作時間とメモリアクセス時間の比 t_a/C_t，すなわち，C_t によってノーマライズされたメモリアクセス時間の変化によってどのように処理速度が変化するかを示している．この図より，t_a/C_t の比が10程度までは処理速度比 S_o, S_n が減少するが，それ以後では，あまり変化しないことがわかる．これは，ノイマンコンピュータとデータフローコンピュータのノード実行時間のほとんどが，メモリアクセス時間によって占められ，他の動作時間の影響が相対的に減少し，メモリアクセス回数の差が主にスピードを決めるためである．また，t_a/C_t が4程度では，高速化されたデータフロー処理速度とノイマン処理速度がほぼ同じ値と

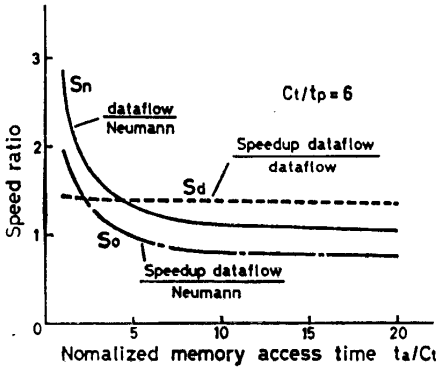


図 7 処理速度比と t_a/C_t の関係
Fig. 7 Speed ratio vs. t_a/C_t.

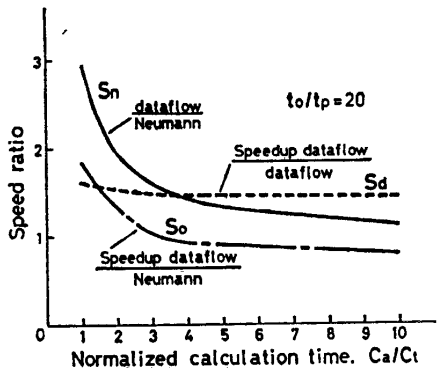


図 8 処理速度比と C_a/C_t の関係
Fig. 8 Speed ratio vs. C_a/C_t.

なり ($S_o=1$), 4以上では，逆に，データフロー処理の高速化が速くなることを表している ($S_o<1$). これらはレジスタ組み合わせ回路とレジスタとの動作時間比 C_t/t_p の値によって著しい変化を示さないので，$C_t/t_p=6$ の場合のみが示されている．図8は計算時間 C_a の処理速度比に与える影響を示す図である．この図より，C_a/C_t が5以上では，ほとんど処理速度比が変化しないことがわかる．この場合も，C_a/C_t が5以上になると，ノードの実行時間のほとんどが計算時間によって占められるようになり，相対的に他の動作時間の影響が少なくなるからである．実際の C_a/C_t の値は5以上であることが多いので，計算時間の処理速度比に与える影響は少ないと考えられる．この特性はメモリとレジスタとの動作時間比 t_o/t_p の値によってほとんど変化しないので，$t_o/t_p=20$ のときの値を代表としてプロットしてある．図7，図8いずれの場合も，高速化まえのデータフローコンピュータとの処理速度比 S_d は約1.4で，t_a, C_a などの影響をあまり受けずに一定であることがわかる．すなわち，本高速化法により，処理速度が約40%程度向上し，この向上率は構成素子の影響をほとんど受けないことを示している．また，高速化したデータフローコンピュータの直列処理速度は，ノイマンコンピュータのそれに匹敵することがわかる．

6. むすび

以上，処理を直列処理と限った場合の，データフローコンピュータの高速化手法とデータフローコンピュータとノイマンコンピュータの処理速度を算出し，その結果を比較した．これらの結果を概念的，具体的に知るために，データフロープログラムにおける命令の分布をギブソン命令ミックスから推定し，直列処理速度の具体的な数値例を求めた．

その結果，高速化以前ではデータフローコンピュー理はノタによる直列処イマンコンピュータのそれに比べて遅いが，高速化後ではほぼ同じになるという一つの結論が得られた．

本論文で，対象としたコンピュータの一方は，現在研究途上のコンピュータであり，他方は，すでに長い研究と実用化の歴史を持つコンピュータであるので，後者では，どのレベルのコンピュータを対象とするかによってその結果が変ってくる．そのため，ここでは，データフローコンピュータの発達段階にノイマンコンピュータをあわせ，できるかぎり基本機能を持ったノイマンコンピュータを対象として，基本的な値を得ようと試みた．実際には，個々のコンピュータによってその能力が違うし，また，データフローコンピュータにも，ノイマンコンピュータで用いられている種々の高速化法が導入されていくと考えられるので，これらの変化に伴って，スピードもある程度変っていくと思われる．しかしながら，ここで得られた結果は両コンピュータの基本的な性能の一つの指標となりうるものであり，今後の研究の基礎データになると思われる．また，一般に，データフロー処理プログラムには，並列処理と直列処理が混在するのが普通であるので，本論文で導入したデータフローコンピュータの直列処理高速化方法は，今後のデータフローコンピュータの高速化に一つの寄与をするものと思われる．本結果をもとに，並列処理をも含めた性能比較が今後の研究に期待される．

参考文献

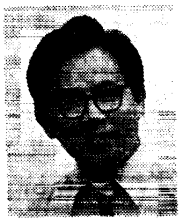
1) Dennis, J.B. and Rong, G.G.: Maximum Pipelining of Aray Operations on Static Data Flow Machine, *Proc. of the IEEE 1983 International Conference on Parallel Processing*, pp. 331-334 (1983).
2) Arvind and Gostelow, K.P.: The U-interpreter, *Computer*, Vol. 15, pp. 42-49 (Feb. 1982).
3) Sowa, M. and Murata, T.: A Data Flow Computer Architecture with Program and Token Memories, *14th Asilomar Conference on Circuits, Systems and Computers* (Nov. 1980).
4) Sowa, M. and Ohokubo, M.: Data-Driven Control of Multi-Microprocessor System, *Trans. of IECE Japan*, Vol. E 67, No. 2, pp. 96-100 (Feb. 1984).
5) 曾和将容：データフローコンピュータによるノイマン処理，信学会電子計算機研資，EC 83-57 (1984. 3).
6) Iwashita, M., Tenma, T., Matsumoto, K. and Kurokawa, H.: Modular Data Flow Image Processor, *IEEE COMPCON Spring '83*, pp. 464-467 (1983).
7) Takahashi, N. and Amamiya, M.: A Data Flow Processor Array System: Design and Analysis, *Proc. of the ACM 1983 International Conference on Computer Architecture*, pp. 243-250 (1983).
8) 大山，浅田，斉藤，猪瀬：環状結合網を用いた分散型データフロー計算機の一方式，信学会論文誌 (D), Vol. J 65-D, No. 12 (1982).
9) 船窪，斎藤，星子：ネットワーク構成データフロー計算機システム，信学会論文誌 (D), Vol.

J 66-D, No. 9 (1983. 9).
10) 島田, 平木, 西田：科学技術計算用データ駆動計算機 Siguma-1 のアーキテクチャ, 信学会電子計算機研資, EC83-20 (1983).
11) 曾和将容：データフローコンピュータにおける直列処理の高速化, 信学会電子計算機研資, EC 83-58 (1984. 3).
12) Farrell, E. P., Grani, N., and Treleavan, P. C.: A Concurrent Computer Architecture and a Ring Based Implementation, *Proc. of IEEE Int. Conf. on Computer Architecture*, pp. 1-11 (1979).
13) Gibson, J. C.: The Gibson Mix, IBM Rep. TR 00. 2043, IBM Corp. Poughkeepsie, N. Y. (June 1970).
14) Gajiski, D. D., Padua, D. A. and Kuck, D. J.: A Second Opinion on Dataflow Machines and Languages, *Computer*, Vol. 15, No. 2, pp. 58-69 (Feb. 1982).

(昭和 60 年 7 月 30 日受付)
(昭和 61 年 2 月 20 日採録)

曾和 将容（正会員）

昭和 49 年, 名古屋大学大学院博士課程電気, 電子専攻修了. 昭和 49 年, 群馬大学工学部情報工学科助手, 昭和 51 年, 助教授, 現在にいたる. この間, 並列処理, コンピュータアーキテクチャ, 特に, データフローコンピュータ, コントロールコンピュータなど次世代汎用コンピュータの研究に従事. 工学博士. IEEE, ACM, 電子通信学会各会員.



羅 四維

1968 年中華人民共和国科学技術大学卒業. 1968 年～1975 年遼寧朝陽電子技術応用研究所勤務, 1975 年北京市北方交通大学電子工程系講師, 1982 年～1984 年群馬大学客員研究員（工学部情報工学科）, 現在, 北京市北方交通大学計算機科学技術系講師. この間, 電子技術応用, 工業自動制御, 論理回路の設計, マイクロプログラムの設計, 計算機アーキテクチャなどの研究に従事.