知能リンクにおけるプログラムのマトリクス表現

Matrix Representation of Programs within An Intelligent Link

宮島 潤　遠藤 昇　中村 維男　重井 芳治
Jun Miyajima　Noboru Endo　Tadao Nakamura　Yoshiharu Shigei
東北大学　工学部
Faculty of Engineering, Tohoku University

1. はじめに

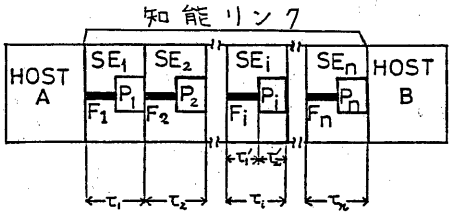
近年、従来のノイマン型計算機の機能の限界が指摘されている。これを打ち破るため、並列処理の概念を基本とした計算機システムの研究が盛んである。その中で"通信"の概念と"処理"の概念を融合し、通信回線に処理を行なう機能を持たせた知能リンクが提案されている[1]。本知能リンクは、光ファイバとプロセッサから成るシステム要素（セグメント）をn個接続し、任意のデータをパイプライン処理しながら伝送することを可能にする。その際、データに先行してプログラムを転送しなければならないが、プログラムの表現方法が適切でないと転送に時間がかかり、全体の処理時間が大きくなってしまう。そこで本稿では知能リンクにおけるプログラムの表現方法について考えることにする。

まず知能リンクシステムについて、システムの概要とソフトウェア体系を説明する。次に生成プログラムと、マトリクスを使用した表現法について考察する。

2. 知能リンクシステム

2.1 知能リンク

知能リンクは図1に示すように、伝送を行なう光ファイバと処理を行なうプロセッサから成るシステム要素を一次元にn段並べて構成される。知能リンクは本来ホスト計算機間における



SEi : i段目のシステム要素
n : システム要素数
Fi : システム要素内の光ファイバ
Pi : システム要素内のプロセサ
τi : システム要素における処理時間
τ1' : 光ファイバ伝送遅延時間
τ2' : プロセサによる処理時間

図1 知能リンクのシステム構成例

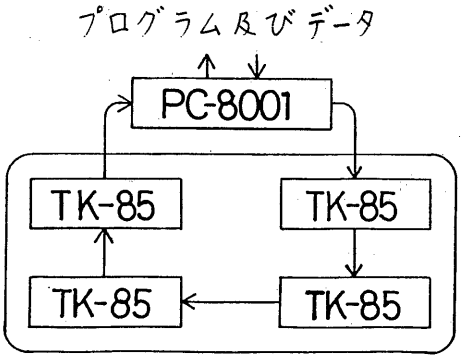


図2 ループ状計算機網の構成

処理を伴った伝送システムであり、処理方式はパイプライン処理である。この際データの処理はデータに先行して供給されるプログラムによって行なわれる。従って知能リンクはソフトウェアによって機能を供給することから汎用パイプライン計算機とみなすことができる。

2．2　試作システムの概要

知能リンクは本来パイプライン処理システムである。その各段の処理規模には限度があるのが普通であるため、規模の大きいジョブをシステム内で終了させるためには多くのプロセス要素を必要とする。しかし実際にはプロセッサ数に限りがあるため、システムとしてはループ状の計算機網の形態をとることにした。

試作システムではマイクロプロセッサ（ＴＫ－８５）４個、入出力の制御とパケット作製のためのホスト計算機としてマイクロコンピュータ（ＰＣ－８００１）１台を用い、図２のようなループ状計算機網を構成した。各プロセッサ間のデータ及びプログラムの送受信にはＡＣＩＡを使用し、８bitずつシリアルに行なっている。この送受信はマイクロプロセッサの割り込み処理で非同期に行なわれる。

2．3　ソフトウェア体系

本処理システムで処理されるソース・プログラムは高級言語の形でホスト計算機に入力されコンパイルされる。そして、その出力コードと処理されるべきデータからパケットを作製し、マイクロプロセッサに転送する。マイクロプロセッサでは目的コードを一つずつ解釈し、データに対する処理を行なう。従って言語体系としてはコンパイラ・インタープリタ方式となっている。

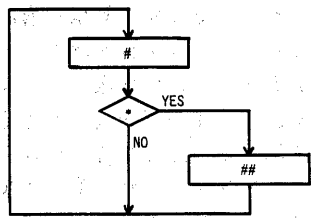
2．4　生成プログラムとその処理

図３に生成プログラムの例を示す。このプログラムは１から10までの和を求めるプログラムである。右側がソース・プログラム、左側がその生成プログラムである。各システム要素での処理を軽減するため、生成プログラムはスタック・マシンを想定した中間言語となっている。この中間言語は逆ポーランド記法をもとにしたもので、演算子及びデータ・アドレスは各々８bitで表わされる。

```
                                                    source program
                                                    CONST n=10;
                                                    VAR i,s;
address  machine code    mnemonic code
  0000   020000          PUSH    0000H             BEGIN
  0003   0702            POP     S                   s:=0;
                 ;
  0005   020100          PUSH    0001H               i:=1;
  0008   0701            POP     I
                 ;
  000A   0801    while    :PUSH   I                 WHILE i <= n DO
  000C   0800             PUSH    N
  000E   15               SUB
  000F   231200           JPP     end(0012H)
  0012   0802             PUSH    S                   BEGIN
  0014   0801             PUSH    I                     s:=s+i;
  0016   14               ADD
  0017   0702             POP     S
  0019   0801             PUSH    I                     i:=i+1;
  001B   020100           PUSH    0001H
  001E   14               ADD
  001F   0701             POP     I
  0021   241A00           JP      while(001AH)        END
                 ;
  0024   2E        end  :END                         END.
                 ;
                 ; DATA AREA
                 ;
  0024   0A00    ; N
  0026   0000    ; I
  0028   0000    ; S

  002A   4040    ; PACKET END
```

図３　生成プログラムの例



•：Arrival of subsequent packet while a current program runs.

#：Processing of one statement ended by pop symbol.

##：Receiving a new packet sent from the previous segment after sending the current packet to the next segment.

図４　各セグメントでの処理方式

各マイクロプロセッサでは原則としてそのパケットの生成プログラムの１ブロックを処理し、次のマイクロプロセッサに転送し、次のパケットの処理に移る。しかしマイクロプロセッサ間のパケット転送はＡＣＩＡを用いてシリアルに行っているため、次のパケットが到着するまでにアイドル・タイムが生ずることがある。これを避けるため、次のパケットが到着するまでは今あるパケットの処理を続けることにした。これを図４に示す。従って生成プログラムの各ブロックがどのマイクロプロセッサで処理される

がは、そのときの状況によって異なる。また処理の終了したパケットはプログラム部が消去されホスト計算機まで転送される。

次に各セグメントにおける処理時間の割合について考察する。例として次のようなプログラムの処理を考える。

$$\begin{cases} Y = A * X + B \\ Z = A * Y + X \end{cases}$$

このプログラムから作製されるパケットは8bit×38の長さとなる。これと同じ内容のパケットを複数個ホスト計算機から次々に送り出すものとする。一つのパケットの受信が完了してから次のパケットの受信が完了するまでを全体の処理時間として、各処理時間とその比率を求めたのが表1である。

	処理時間	比率
受信処理	4416	10.4
送信処理	4647	11.0
パケット処理	1984	4.7
アイドル・タイム	31300	73.9
計	42347(μs)	100.0(%)

(ACIAのクロック 9.6 KHz)

表1 各処理時間とその比率

表1からわかるように、全体の処理時間のうちパケットの処理に使われているのは5%程度である。プログラムの中にループを含むようなものであれば、この比率を高くすることはできるが全体の処理時間は減らすことができない。しかし少ないbit数でより多くのプログラムを表現することができれば全体の処理時間を減らすことができる。次章では、その方法について考察する。

3. プログラムのマトリクス表現

3.1 マトリクスの性質

ここでプログラム内の演算の遷移を表わすのに遷移行列を用いる方法について考察する。遷移行列の要素 α_{ij} は i から j への遷移があるときは"1"、ないときは"0"である[3]。

プログラム内の演算の遷移が同一の遷移を繰り返しているときは遷移行列は一種類でよい。しかしこのようなことは実際にはまれであり、遷移のパターンが変わった場合には新たな遷移行列が必要となる。新たな遷移行列をそのまま与える方法もあるが、ここでは行の入れ換えと列の入れ換えで新たな遷移行列を生成する方法について考察する[4][5]。

$N \times N$ の行列を考える。この行列に対して、行の入れ換えは $N!$ 通り、列の入れ換えも $N!$ 通りあるので、行の入れ換えと列の入れ換えによって生ずる行列は $(N!)^2$ 通り存在する。従って新しい遷移行列を一つ生成するには $\lceil \log_2 (N!)^2 \rceil$ (bit)必要である。$N \times N$ の行列が M 個必要なときには元になる行列を転送し、$(M-1)$ 回新たな行列を生成させればよいから必要なbit数 P は

$$P = N^2 + \lceil \log_2 (N!)^2 \rceil \times (M-1) \quad (1)$$

となる。これに対して $N \times N$ の行列 M 個をそのまま転送する場合には

$$F = MN^2 \ (\text{bit}) \quad (2)$$

必要となる。この両者の関係を表わしたのが図5と図6である。図より M, N が大きい程、行

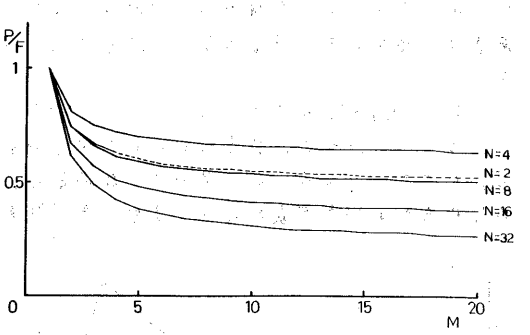


図5 必要なbit数の比較①

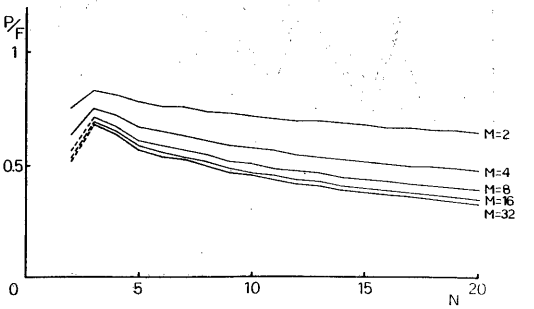


図6 必要なbit数の比較②

の入れ換えと列の入れ換えをしたほうが少ないbit数で済むことがわかる。

3．2　生成プログラムのマトリクス表現

生成プログラムは逆ポーランド記法をもとにしたものであり、演算子及びデータ・アドレスは各々8 bitで表わされている。まずこの生成プログラムの中から演算子だけを抜き出す。この際データ・アドレスを表わしていた8 bitのうちの余っているbitを使用して、どこに演算子があったかがわかるようにしておく。次に抜き出した演算子の遷移を遷移行列に表わす。途中で演算子の遷移のパターンが変わったときには3.1で述べたように行の入れ換えと列の入れ換えで新たな遷移行列を与えることにする。

ここで従来の方法との比較するため、N種類の演算子を使用している場合を考える。N×Nの遷移行列を使って最も効率よくプログラムが表わせた場合について従来の方法と比較する。このとき一つの遷移行列で表わせる最大の演算数はNであるから、プログラムがN×Nの遷移行列M個で表わせたとすると、必要なbit数Pは式(1)になる。また全体の演算の数はN×Mであるから従来の方法で表わすのに必要なbit数Fは

$$F = \lceil \log_2 N \rceil \times N \times M \qquad (3)$$

となる。この両者の比を表わしたのが図7である。Mが大きくなるに従って行の入れ換えと列の入れ換えの効果が表われてくるが、まだ従来の方法のほうが優れている。

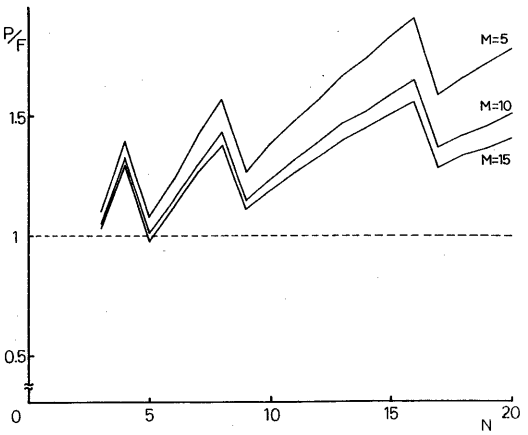


図7　従来の方法との比較①

次に行の入れ換えと列の入れ換えを使用しない方法について考察する。簡単な例として、演算が4種類（＋，－，＊，↑）の場合について考える。遷移行列の大きさはN×Nであり、遷移行列の一例を図8に示す。この遷移行列の中

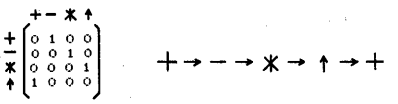


図8　遷移行列の例

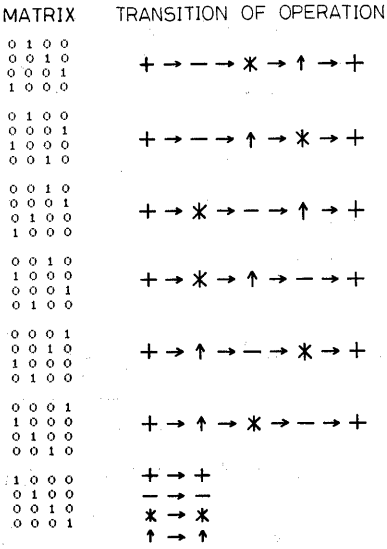
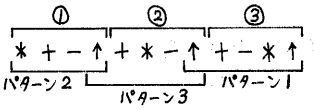


図9　遷移行列のパターン

で"＋"から始まり残り3種類の演算をすべて1回ずつ遷移し"＋"に戻ってくるようなものは3!（＝6）通りある。これに単位行列を加えた計7種類（図9参照）の行列で演算の遷移を表わすことにする。以下具体的な例で考える。

$$\begin{cases} X = A * B + C - D & ① \\ Y = (B + C) * A - D & ② \\ Z = (A + B - C) * D & ③ \end{cases}$$

これらの式から演算の遷移だけを抜き出すと下のようになり、3個の行列で表わせる。



これを　(1)マトリクス内のスタート位置　(2)マ

トリクスのパターンの番号　(3)遷移回数　以下(2)と(3)の繰り返し　の順で表わしていく。この方法で①～③を表わすのに必要なbit数Lは

$$L=\lceil \log_2 4\rceil+(\lceil \log_2\{(4-1)!+1\}\rceil+\lceil \log_2 4\rceil)\times 3$$
$$=17\ (\text{bit})$$

これに対して従来の方法では

$$F=\lceil \log_2 4\rceil\times 12$$
$$=24\ (\text{bit})$$

必要となる。

次に演算がN種類のときについて従来の方法とbit数を比較する。プログラム内の演算の数がM個であるとし、それがN×Nの遷移行列A個で表わせたと仮定する。ただし一つの行列で表わせる最大の遷移回数はNであるから

$$A\geqq\lceil (M-1)/N\rceil$$

となる。具体例のときと同じようにしてL，Fを求めると

$$L=\lceil \log_2 N\rceil+(\lceil \log_2\{(N-1)!+1\}\rceil+\lceil \log_2 4\rceil)\times A \quad (4)$$

$$F=\lceil \log_2 N\rceil\times M \quad (5)$$

となる。M＝100の場合についてLとFの比を求めたのが図10である。

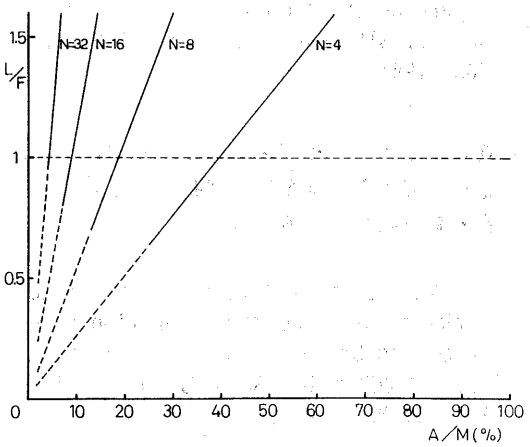


図10　従来の方法との比較②

図10より、少ない数の遷移行列で表わせるようなプログラムに対してはこの方法が有効であることがわかる。実際のプログラムでは異なる種類の演算を次々と行なうものに対して有効であることになる。しかしこの方法でもNが大きくなると有効でなくなるので、遷移行列そのものを少ないbit数で表わす方法について次節では考察する。

3．3　マトリクスの圧縮

今まで考えてきた遷移行列は非常に"0"の要素数が多い。遷移行列の中で条件分岐を扱ったとしても、1行の中の"0"の要素は高々2個である。そこで次の3種類の方法でマトリクスを表わす方法について、直接マトリクスを表わす方法との比較を行なった。ただしN×N行列を考え、A，B，Cを次のように定める。

A：＂0＂の要素のみからなる行の行数
B：＂1＂の要素が1個だけからなる行の行数
C：＂1＂の要素が2個だけからなる行の行数

従ってN＝A＋B＋Cとなる。

(i)① ＂0＂のみからなる行の行数を表わすのに$\lceil \log_2 N\rceil$ (bit)
＂0＂のみからなる行を指定するのに
$\lceil \log_2 N\rceil\times A$ (bit)
② ＂1＂が1個だけからなる行の行数を表わすのに$\lceil \log_2 N\rceil$ (bit)
＂1＂が1個だけからなる行を指定するのに$\lceil \log_2 N\rceil\times B$ (bit)
＂1＂の要素がある位置を指定するのに
$\lceil \log_2 N\rceil\times B$ (bit)
③ ＂1＂が2個だけからなる行の＂1＂の要素の位置を指定するのに
$\lceil \log_2 N\rceil\times 2\times C$ (bit)

(ii)① ＂0＂のみからなる行の指定にN (bit)
② ＂1＂が1個だけからなる行の指定にN (bit)
＂1＂の要素の位置の指定に
$\lceil \log_2 N\rceil\times B$ (bit)
③ ＂1＂が2個だけからなる行の＂1＂の位置の指定に
$\lceil \log_2 N\rceil\times 2\times C$ (bit)

(iii) 行のパターンが全部で何通りあるかを考える。
① ＂0＂のみからなる行は1通り

② "1"が1個だけからなる行はN通り
③ "1"が2個だけからなる行は ${}_NC_2$ 通り

以上①～③の計 $1+N+{}_NC_2$ （通り）の中から一つのパターンを指定する。

以上3種類の方法でマトリクスを表わすとき必要となるbit数はそれぞれ次のようになる。

(i) $L1=\lceil \log_2 N \rceil \times (A+2B+2C+2)$

(ii) $L2=\lceil \log_2 N \rceil \times (B+2C)+2N$

(iii) $L3=\lceil \log_2 (1+N+{}_NC_2) \rceil \times N$

L1～L3とマトリクスを直接表わす場合に必要となるbit数F（$F=N^2$）との比を表わしたのが図11、図12である。

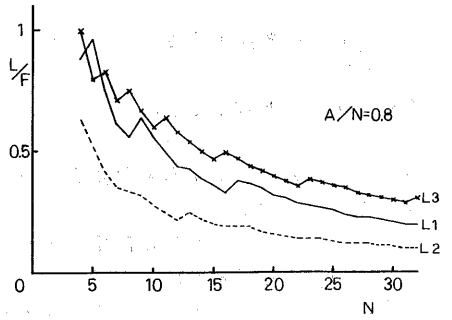


図11 マトリクスの圧縮比①

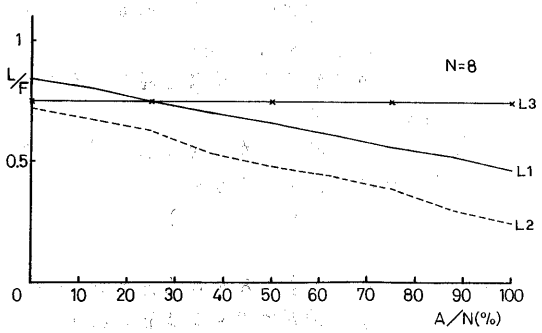


図12 マトリクスの圧縮比②

図よりL1～L3ともNが大きい程、またL1とL2は、"0"のみからなる行が多い程、効果が大きいことがわかる。

4 まとめ

知能リンクにおける生成プログラムの転送時間を短縮する必要性を説明し、それに関連してプログラムを表わすのに必要なbit数を少なくする方法の一例として、生成プログラムのマトリクス表現について考察した。

演算の遷移のみをそのまま遷移行列で表わす方法ではあまり大きな効果が期待できないので、今後は使用するデータの遷移も同時に一つの遷移行列で表わす方向で考えて行きたい。

参考文献

(1) T.Nakamura and Y.Shigei,"Proposal of an Intelligent Link Applied to Computer Networks," Pacific Telecommunication Conference Proc., pp.2B-13-2B-20,Jan., 1980.

(2)遠藤、中村、重井 "マイクロプロセッサを用いた知能リンクの構成について" 昭和57年電子通信学会総合全国大会

(3) T.Nakamura and Y.Shigei,"Developing Communications Software for An Intelligent Link," IEEE International Coference on Communications. Coference Record. 81CH1648-5, pp. 53.3.1-53.3.5, June 1981.

(4)遠藤、中村、重井 "知能リンクのプログラム構成に関する一考察"，昭和56，情報理論とその応用研究会，G－3

(5) T.Nakamura and Y.Shigei,"An Intelligent Link-Oriented Program Generation Under Permuting States of Markov Chains," IEEE GROBECOM'82, Record, pp.E3.1.1-E3.1.5, Nov.-Dec., 1982.