設計自動化16－1
（1983. 2. 15）

コンピュータネットワークによる並列論理シミュレーションの検討

古賀義亮　竹之上典昭
（防衛大学校・電気工学教室）

1．まえがき

ＶＬＳＩの発達により回路の集積化が進み、論理シミュレーションの実行に多大な時間を費やすようになった。将来さらに回路の集積化が進めば一台のコンピュータによる論理シミュレーションは実行不可能な状態にまでなるであろうといわれている。論理シミュレーションの高速化を図るには、一台のコンピュータの処理速度を高速化するだけに留まらず、複数のコンピュータを使用する並列処理も併せて開発すべきである。すでにＩＢＭから、論理シミュレーションを並列処理するＹｏｒｋｔｏｗｎ Ｓｉｍｕｌａｔｉｏｎ Ｅｎｇｉｎｅ（ＹＳＥ）という専用機が出現しており、そのことからも並列処理の必要性は明らかである。本報告では、汎用の3ポートコンピュータネットワークを用いて並列に論理シミュレーション（ゲートレベル）を行なう方法について提案を行ない、その検討を行なう。さらに、ＭＯＳレベルのシミュレーションについても新たに提案を行なう。

2．コンピュータネットワーク

本報告で取り扱うコンピュータネットワークは、各ノードコンピュータが、3本の入出力端子を持つ3ポートコンピュータネットワークである。図2－1は、3ポートコンピュータのネットワーク例であり、各ノードコンピュータには、表2－1のような連結表を持たせる。コンピュータネットワークを用いて、論理シミュレーションを並列処理するために、論理回路の各素子をノードコンピュータに割り振って処理する方法を用いる。この方法で並列処理するためには、次の条件が必要である。

（1）各ノードコンピュータに割り振られた素子は、ほぼ均等に配分する。

（2）ノードコンピュータの数がネットワークの大きさにより限定されるため、シミュレーションする回路の素子数が多い場合、一つのノードコンピュータに多数の素子に関する情報を蓄積することができる。

（3）論理回路をコンピュータネットワーク上に展開し、各ノードコンピュータに蓄積するためには、ネットワーク上のいずれかのノードコンピュータから入力

表2－1　連結表

ノード	0	1	2
0	1	2	15
1	3	4	0
2	0	5	6
3	7	1	16
4	1	8	5
5	4	10	2
6	2	11	19
7	3	12	8
8	7	9	4
9	8	13	10
10	9	11	5
11	10	14	6
12	7	17	13
13	12	14	9
14	13	18	11
15	16	0	19
16	17	3	15
17	16	18	12
18	17	19	14
19	18	6	15

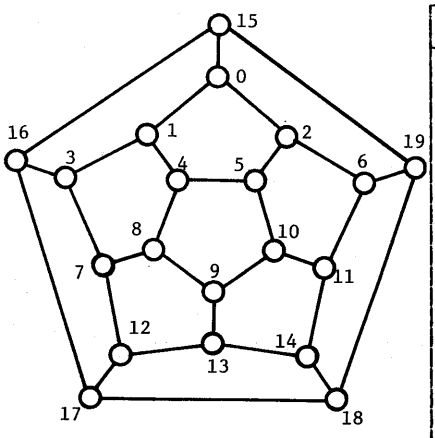

図2－1　3ポートコンピュータネットワーク

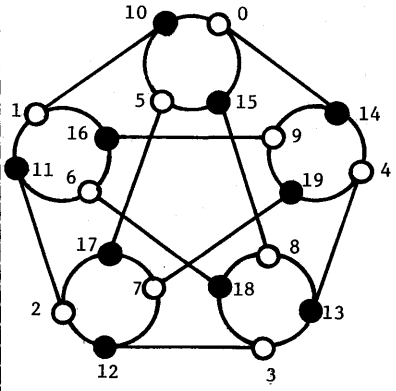

図2－2　AONの一例

可能とする。

（４）各ノードコンピュータ相互間のデータ送受信においてデータの衝突を起こさない。

又、論理回路は２分木構造に変換（５章参照）して展開するので、完全２分木をコンピュータネットワーク上に展開すると、図２－１のネットワークでは表２－３のようになる。この表で＊は、そのレベルでの展開動作が一回以上行なわれたことを示す。表内の数は、そのレベルまでにノードコンピュータに蓄積された素子の量を示している。表２－３では、レベル７以上になるとすべてのノードコンピュータが展開動作を行なっている。この状態では、ノードコンピュータは３方向同時にデータの送受信を行なう必要がある。　図２－１のネットワークで、データを衝突させないようにするためには、ノードコンピュータ間の通信のプロトコルによる制御等を用いなければならない。しかし、図２－２の３ポートネットワークで同様に完全２分木を展開すると表２－４のような結果が得られる。この表の＊に注目すると奇数レベルでは右半分に表われ、偶数レベルでは左半分に表われている。また、蓄積量も＊が表われたノードコンピュータだけが増加している。　これはコンピュータネットワークが交互に動作していることを示している。このような交互動作を行なうネットワークをＡＯＮ（Ａｌｔｅｒｎａｔｅ Ｏｐｅｒａｂｌｅ Ｎｅｔｗｏｒｋ）と呼ぶ。ＡＯＮは偶数個のノードコンピュータから成り、図２－２のように２つのグループ（白ノードと黒ノード）に分割すると、互いに反対のグループのノードと結合するような連結２部グラフのコンピュータネットワークである。この特性により、２つのグループは交互に送受信を行なうことができ、データの衝突を防ぐことができる。図２－３はノードコンピュータ１２０個のＡＯＮ上に完全２分木を展開した場合の蓄積量を隠線消去の手法を用いて３次元表示したもので、中央を縦に走る断層面が、ＡＯＮが交互に

表2－2　AONの連結表

ノード	0	1	2
0	10	15	14
1	11	16	10
2	12	17	11
3	13	18	12
4	14	19	13
5	15	10	17
6	16	11	18
7	17	12	19
8	18	13	15
9	19	14	16
10	5	0	1
11	6	1	2
12	7	2	3
13	8	3	4
14	9	4	0
15	0	5	8
16	1	6	9
17	2	7	5
18	3	8	6
19	4	9	7

表2－3＊＊　2ﾌﾞﾝｷﾞ　ノ　ﾃﾝｶｲ　3PORT NETWORK＊＊

\node / Level\	N0	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10	N11	N12	N13	N14	N15	N16	N17	N18	N19	
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	: 3
2	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	: 7
3	1	1	1	1	2	2	1	1	1	0	1	1	0	0	0	0	1	0	0	1	: 15
4	1	2	2	1	2	2	1	2	3	2	3	2	1	0	1	2	1	1	1	1	: 31
5	5	2	2	3	3	3	3	3	4	6	4	3	3	4	3	2	2	3	3	2	: 63
6	5	7	7	5	5	5	5	6	6	8	6	6	8	12	8	4	6	6	6	6	: 127
7	9	11	11	13	12	12	13	13	11	12	11	13	18	16	18	12	11	14	14	11	: 255
8	25	20	20	25	24	24	25	30	25	22	25	30	27	24	27	22	27	31	31	27	: 511
9	45	52	52	51	45	45	51	51	54	50	54	51	49	44	49	54	56	57	57	56	: 1023
10	109	101	101	109	103	103	109	94	96	108	96	94	98	100	98	112	108	100	100	108	: 2047

TI$=000341

動作していることを表わしている。　本報告では、ＡＯＮを用いることによりコンピュータネットワーク上での並列論理シミュレーションを円滑にしかも高速に行なう方式を用いる。

３．　ノードコンピュータの構造

ＡＯＮを構成するノードコンピュータの構造について述べる。本方式によるシミュレーションでは、ＡＯＮの特性上からノードコンピュータには、次の能力が要求される。

（１）３方向に対してデータの送受信ができること。

（２）送受信と並列してデータ処理（データフローによる実行の制御）を行なえること。

（３）配分された各素子に関する情報を一時的に蓄積できること。

この要求に答えるため次の２つの構造について検討する。

３．１　１－ＣＰＵ＆３－バッファ

図３－１がこのノードコンピュータの構造である。　このノードコンピュータは、１つのＣＰＵと３つのバッファで構成され、すべての処理は１つのＣＰＵによってなされる。３つのポートに対するデータの送受信のため各ポートにはバッファが必要である。３方向から同時に受信したデータの処理は１つのＣＰＵで逐次に処理する。

３．２　３－ＣＰＵの結合

図３－２が３つのＣＰＵを使用したノードコンピュータの

表2－4　** 2ﾌﾞﾝｷﾞ ﾉ ﾃﾝｶｲ　AON**

\node / Level\	N0	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10	N11	N12	N13	N14	N15	N16	N17	N18	N19	
1	↕ 1	0	0	0	0	0	0	0	0	0	↕ 1	0	0	0	0	↕ 1	0	0	0	0	: 3
2	↕ 1	↕ 1	0	0	0	↕ 2	0	0	↕ 1	0	1	0	0	0	0	1	0	0	0	0	: 7
3	1	1	0	0	0	2	0	0	1	0	↕ 2	↕ 1	0	↕ 1	0	↕ 2	↕ 1	↕ 2	↕ 1	0	: 15
4	↕ 3	↕ 2	↕ 3	↕ 2	↕ 1	2	↕ 3	↕ 2	↕ 2	↕ 1	2	1	0	1	0	2	1	2	1	0	: 31
5	3	2	3	2	1	2	3	2	2	1	↕ 3	↕ 6	↕ 7	↕ 3	↕ 4	↕ 3	↕ 4	↕ 3	↕ 5	↕ 4	: 63
6	↕ 5	↕ 9	↕ 10	↕ 11	↕ 9	↕ 5	↕ 9	↕ 10	↕ 7	↕ 10	3	6	7	3	4	3	4	3	5	4	: 127
7	5	9	10	11	9	5	9	10	7	10	↕ 13	↕ 16	↕ 17	↕ 21	↕ 15	↕ 11	↕ 20	↕ 19	↕ 17	↕ 21	: 255
8	↕ 30	↕ 31	↕ 32	↕ 33	↕ 39	↕ 33	↕ 35	↕ 37	↕ 35	↕ 36	13	16	17	21	15	11	20	19	17	21	: 511
9	30	31	32	33	39	33	35	37	35	36	↕ 68	↕ 66	↕ 68	↕ 65	↕ 74	↕ 76	↕ 62	↕ 64	↕ 69	↕ 70	: 1023
10	↕ 159	↕ 134	↕ 134	↕ 136	↕ 131	↕ 142	↕ 127	↕ 128	↕ 140	↕ 134	68	66	68	65	74	76	62	64	69	70	: 2047

TI$=000340

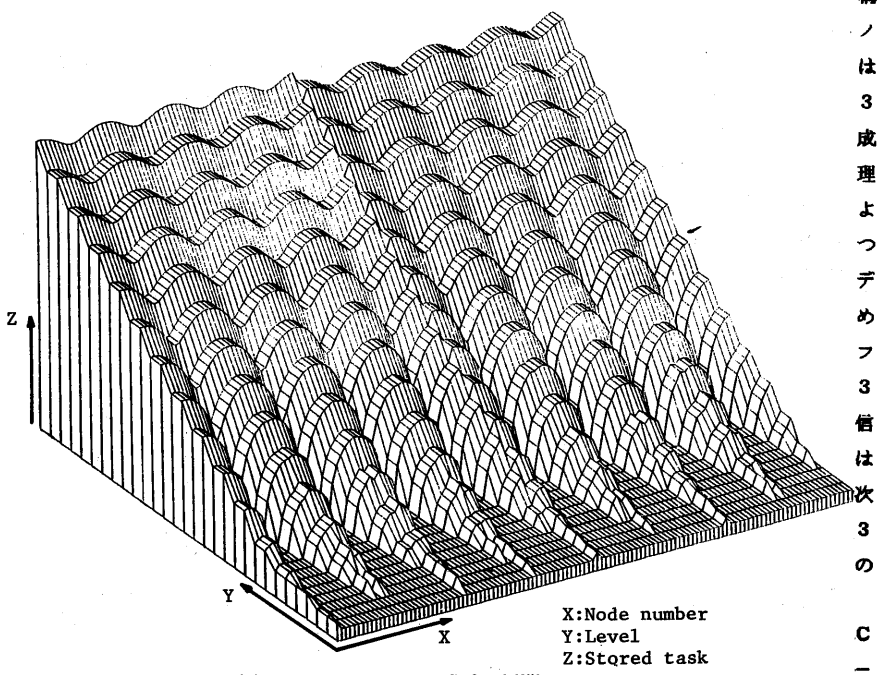

図2－3　ＡＯＮでの2分木の展開

構造である。　各ＣＰＵ（ｘ，ｙ，ｚ）は、それぞれ一つづつのポートを担当しており、各ＣＰＵは図３－３のように共通メモリによって結合している。この構造では、各ＣＰＵは、それぞれの担当ポートのみに対してのデータの送受信及び処理を実施する。

図３－１の構造と図３－２の構造を比較すると、並列処理を行なうためには、図３－２の構造の方が有利である。本来並列に処理されるべき３方向からのデータを図３－１の構造の場合では、逐次に処理しなければならないが、図３－２の構造では、３つのノードコンピュータで並列に処理できる。さらに図３－２の構造をＡＯＮの特徴とあわせ考えれば、各ポートには必らずしもバッファを設ける必要がないことがわかる。つまり、各ＣＰＵは、自分のメモリにデータを書きこむのと同じように隣のノードコンピュータにデータを送ることができる。以上のことからここでは図３－２の構造のノードコンピュータを採用し、ＴＮＣ（Ｔｈｒｅｅ　ｐｏｒｔ　Ｎｏｄｅ　Ｃｏｍｐｕｔｅｒ）と呼ぶこととする。

４．　回路記述言語

コンピュータネットワークを用いて並列論理シミュレーションを行なうためには論理回路をコンピュータに入力する必要がある。又、並列処理の実行を行なうためにも新しい回路記述言語が必要となる。そこで、次の点に注目した回路記述言語を提案する。

（１）回路記述にあたり物理的な意味との対応を付けやすい表現にする。
（２）並列処理の記述が容易である。
（３）構造化言語の体系を用い階層的な記述を可能にする。

回路の記述は、ＰＡＳＣＡＬの手続きのような構造と役割をもつ回路文（ｃｉｒｃｕｉｔ）　という手続きによって行なう。図４－１－ａのような直列加算回路を例とすると、直列加算回路は、全加算器、レジスタなどの回路で構成されている。また、直列加算回路の部品としての全加算器も半加算器やその他の素子によって構成されている。このような階層構造を回路文で記述すると、図４－２のようになる。この図では、ＲＥＧＩＳＴＥＲやＦＵＬＬＡＤＤＥＲは、ＳＩＲＩＡＬＡＤＤＥＲの中のみで有効である。また、論理回路の物理的な意味と対応づけるため、入出力端子（ｐｏｒｔ）、接続端子（ｔｅｒｍｉｎａｌ）、接続線（ｌｉｎｅ）、部品の関数機能（ｅｌｅｍｅｎｔ．．．ｆｕｎｃｔｉｏｎ）、端子の接続（ｃｏｎｎｅｃｔｉｏｎ）などの宣言文がある。また、並列処理を指示する（ｐａｒａｌｌｅｌ）がある。

図４－３は集積化論理演算装置（ＡＬＵ）を記述した例である。　ｖａｒ　からｆｕｎｃｔｉｏｎ　の終わりまでが宣言部分であり、ｃｉｒｃｕｉｔ　ＳＵＢは、　ｃｉｒｃｕｉｔ　ＡＬＵ　の中の

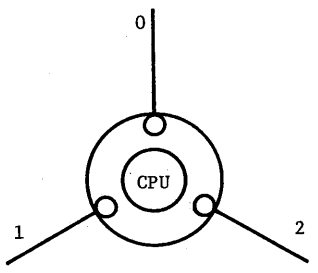

図３－１　１－ＣＰＵ＆３－バッファ

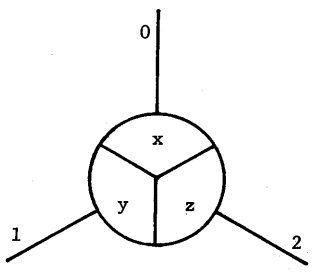

図３－２　３－ＣＰＵの結合（ＴＮＣ）

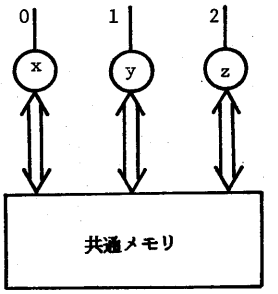

図３－３　ＴＮＣ（メモリとの結合）

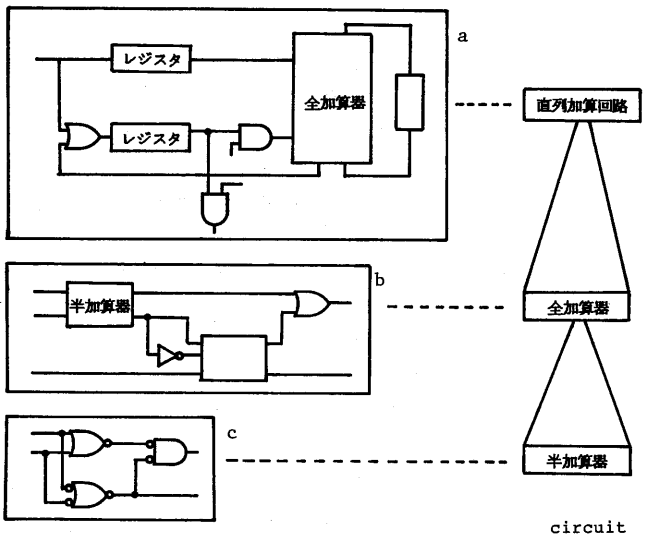

図4－1　回路の階層構造

```
circuit SERIALADDER;
  circuit REGISTER;
  end;
  circuit FULLADDER;
    circuit HALFADDER;
    end;
  end;
end;
```

図4－2　回路文の構造

サブシステムとしての回路である。そして、実行部分は、ｂｅｇｉｎ　ｐａｒａｌｌｅｌ　から　ｅｎｄ　の間である。

このｐａｒａｌｌｅｌ　により、ＡＯＮによる並列論理シミュレーションを指示している。

５．　回路のネットワークへの展開とシミュレーションの実行

論理回路をコンピュータネットワーク上に展開する方法について図５－１の回路を例として説明する。これは排他的論理和の回路である。この回路の各素子をノードとする木として表現すると図５－２のようになる。

```
circuit  ALU;
  var J;
  port
    A,B,S,F:array[0-3];
    M,CN,GY,CN4,PX,AB;
  line
    LI1,LI2:array[0-3];
  element
    AND3(I1,I2,I3);
    AND4(I1,I2,I3,I4);
    AND5(I1,I2,I3,I4,I5);
    NOR2(I1,I2);
    NOR3(I1,I2,I3);
    NOR4(I1,I2,I3,I4);
  function
    AND3:=I1 and I2 and I3;
    AND4:=I1 and I2 and I3 and I4;
    AND5:=I1 and I2 and I3 and I4 and I5;
    NOR2:=not(I1 or I2);
    NOR3:=not(I1 or I2 or I3);
    NOR4:=not(I1 or I2 or I3 or I4);
  circuit SUB(A,B,P,Q);
    line LNOT,F1,F2,G1,G2,G3;
    begin
      LNOT:=not B;
      F1:=B and S[3] and A;
      F2:=A and S[2] and LNOT;
      G1:=LNOT and S[1];
      G2:=S[0] and B;
      G3:=pass A;
      P:=not(F1 or F2);
      Q:=not(G1 or G2 or G3)
    end;
  begin  parallel
    for J:=0 to 3 do
      SUB(A[J],B[J],LI1[J],LI2[J]);
    GY:=NOR4(pass(LI2[3]),(LI1[3] and LI2[2]),
        AND3(LI1[3],LI1[2],LI2[1]),AND4(LI1[3],
        LI1[2],LI1[1],LI2[0]));
    CN4:=not(GY) or not(not(AND5(LI1[3],LI1[2],
         LI1[1],LI1[0],CN)));
    PX:=not(AND4(LI1[3],LI1[2],LI1[1],LI1[0]));
    F[3]:=(LI1[3] xor LI2[3]) xor NOR4(AND5(
          CN,LI1[0],LI1[1],LI1[2],M),AND4(LI1[1],
          LI1[2],LI2[0],M),AND3(LI1[2],LI2[1],M),
          (LI2[2] and M));
    F[2]:=(LI1[2] xor LI2[2]) xor NOR3(AND4(CN,
          LI1[0],LI1[1],M),AND3(LI1[1],LI2[0],M),
          (LI2[1] and M));
    F[1]:=(LI1[1] xor LI2[1]) xor NOR2(AND3(CN,
          LI1[0],M),(LI2[0] and M));
    F[0]:=(LI1[0] xor LI2[0]) xor not(CN and M);
    AB:=AND4(F[3],F[2],F[1],F[0]);
  end;
```

図4－3　ＡＬＵ記述例

図５－２は２分木構造になっているが、一般の論理回路を木として表現しても２分木にはならないことが多い。本報告で用いるノードコンピュータはＴＮＣであるので、木はすべて２分木に変換する。

２分木構造となった論理回路をＡＯＮ上に展開するためには、回路情報をデータ各ＴＮＣへ送り、各素子の機能をタスクとして蓄積する。そのために、この２分木を行きがけ順になぞって一列化を行ない、

f=+(*(!(A).B).*(A.!(B)))

のような構造に変換する。各ＴＮＣはこれを２方向へ分割しながら展開する。図５－３に展開の様子を示す。各ＴＮＣは先頭に出てくる演算子をタスクとして取り込み、その後に続くデータを左右に分割して次のＴＮＣに送っている。

各ＴＮＣに取り込まれたタスクは、図５－４のような構成で与えられ、親のタスクのデータを指し示すポインタ（ＬＯＣ）と、データを入れる箱（ＤＡＴＡ）をもつ。そして、各ＤＡＴＡという箱にはさらにデータが有意となることを示すトークンを入れる場所がある。

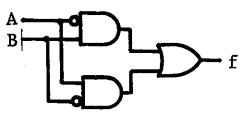

図５－１　排他的論理和

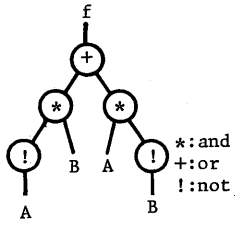

図５－２　木としての表現

ＡＯＮ上へ展開された論理回路は展開の逆順、つまり木で表わすと葉となるタスクからデータを流してシミュレーションを行なう。この場合、蓄積されたタスクのうち、どのタスクを実行するかという判断は、データの流れを示すトークンによって与え、トークンのそろったタスクから実行する。以上の方法により、ＡＯＮ上をデータは衝突することなくスムーズに流れ、シミュレーションが実行される。

６．　並列論理シミュレーションの検討

５章までに述べてきた並列論理シミュレーションの方法についてのシミュレーション結果をもとに、本方式の特徴を明らかにする。そのため、図２－２のＡＯＮ上でＡＬＵの論理シミュレーションを行なう場合について、ＡＯＮの状態及び処理情況をシミュレーションした結果を示す。　このシミュレーションを行なうにあたり、アルゴリズムの開発は、マイコンのＭＺ－８０ＢのＰＡＳＣＡＬを用い、これをもとに実際のシミュレーションにあたっては、計算機にＨＩＴＡＣ　Ｍ２００Ｈを、言語はＰＡＳＣＡＬを用いた。また、ＡＬＵをネットワークへ展開するための、一列化は、マイコンのＰＣ－８８０１を用いて行ない、そのデータはＴＳＳを通じて、ＨＩＴＡＣ　のデ

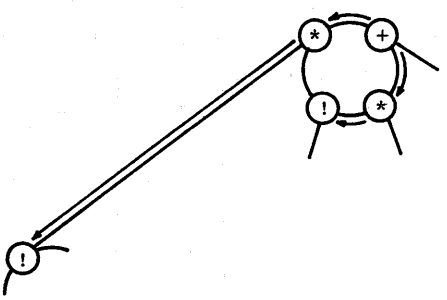

図５－３　木の展開

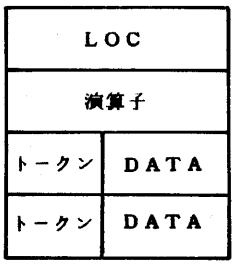

図５－４　タスクの構成

```
GY=nor(11.12.13.14)
CN4=nor(GY.15)
PX=nand(31.33.35.37)
11=pass(32)
12=and(31.34)
  .
  .
  .
```

図６－１　結線情報の記述

ータセットへ送り込むことによって実行した。シミュレーションまでの手順を以下に示す。

（１）ＡＬＵの結線情報を図６－１の様に入力する。

（２）結線情報の演算子を表６－１の対応表に基いて変換する。

（３）結線情報の多入力を２入力に変換する。

（４）根となるべき出力端子を指定して一列化を行なう。

この手順によりＡＬＵの一列化データを作ると、ＡＬＵには、８つの出力端子があるため、８つの木のデータが作製される。さらに、シミュレーションのためには、表２－２のＡＯＮの連結表のデータ、及び、論理シミュレーション用データも入力しなければならない。図６－２はこのシミュレーションに必要なデータ及び出力を図示したものである。

以上の前提にもとづき、３種のＡＬＵ展開法によるシミュレーション結果を以下に示す。

６．１　１つのＴＮＣからの展開

ＡＬＵの８つの木をノード０のポート２のＣＰＵ（以下Ｎ（０，２）と記述する）から展開すると、表６－２のように各ＣＰＵにタスクが蓄積される。Ｐｏｒｔ０，Ｐｏｒｔ１，Ｐｏｒｔ２　の欄がそれであり、ＳＵＭの欄はノード毎の蓄積量、％の欄は全タスクに対するノードへの配分をパーセント表示したものである。また、平均と分散は、ＣＰＵ毎の蓄積量に対するものである。表６－２を見るとＡＬＵが５７５のタスクによって構成されていることがわかる。しかし、入力したＡＬＵの回路素子は６３であり、一台のコンピュータで逐次処理をするためめファンインを考慮しても素子数は１００である。つまりＡＯＮ上に展開したことによって素子数を比較すれば５．７５倍になっている。これは本方式による回路の展開上の欠点である。このように１つのＴＮＣから展開するのを以後ＴＹＰＥ－Ｐということにする。

６．２　それぞれの木をＮ（０，２）からＮ（７，２）に振り分けて展開

次にＡＬＵの８つの木を１つづつのＴＮＣに振り分けて展開する（ＴＹＰＥ－Ｅ）と、表６－３のようになる。こうするとＴＹＰＥ－Ｐと比べて、分散が１９．３１から６．４１へと小さくなり、図６－３の度数分布図を見るとＴＹＰＥ－Ｅの方がＡＯＮにタスクをより均等に配分していることがわかる。この結果、表６－５のＣａｌｃｕｒａｔｅ欄の７番と１４番を比べてもわかるようにＴＹＰＥ－Ｅの方が少ないステップ数でシミュレ

and	:	*
or	:	+
not	:	!
xor	:	@
gate	:	%
pass	:	¥
nand	:	&
nor	:	?
Root	:	#

表６－１　演算子の対応

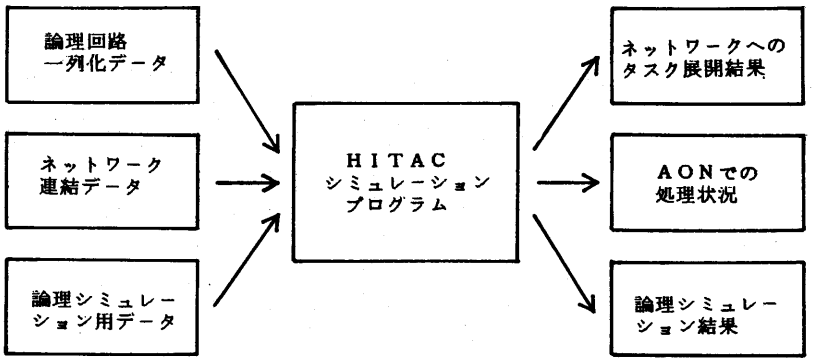

図６－２　シミュレーションに必要なデータ及び出力

ーションを終了している。しかし、この方法もＡＯＮに展開された素子数に関する欠点は同じである。そこでＡＬＵの回路をもう一度見直すために図４－３を見ると以下のことがわかる。　８つの出力端子のうち５つの端子は他の３つの端子に出力するための中間の部品からの出力端子である。つまり、５つの端子を根とする木は、３つの端子を根とする木の部分木になっている。　この特徴を利用すると、３つの木を展開するだけでシミュレーションが行なえる。　この方法を採用するためには、木を一列化する際に、演算子に出力端子記号を付加するだけでよい。出力端子記号を付加されたタスクは、シミュレーション結果を記憶しておくようにする。　シミュレーション終了後、親となるノードコンピュータの呼びかけに応じて結果を回収する。図２－２のＡＯＮであれば、最大５回の交互動作によってすべての結果が回収できる。

６．３　３の木をＮ（０，２）からＮ（２，２）に振り分けて展開

３の木を展開する（ＴＹＰＥ－ＥＳ）と、表６－４のようになり、タスクの総数も３１７に減少する。これは、８つの木を展開する場合に比べて、約４５％の減少であり、処理速度に大きく影響している。表６－５のＣａｌｃｕｒａｔｅをＤａｔａ数で割ると一つの論理シミュレーションのステップ数がわかる。１４番のＴＹＰＥ－Ｅと１５番のＴＹＰＥ－ＥＳの　１－ｉｎｓｔ欄を比べると　３１対１９　になっており、約１．６倍速く

表6－2　ＡＬＵの展開時のタスクの蓄積量（ＴＹＰＥ－Ｐ）
平均　９．５８　　分散　１９．３１

Node	Port 0	Port 1	Port 2	SUM	%
0	8	5	11	24	4
1	10	3	15	28	5
2	11	16	19	46	8
3	8	10	13	31	5
4	10	5	11	26	4
5	11	11	12	34	6
6	12	10	12	34	6
7	17	12	5	34	6
8	7	3	13	23	4
9	3	9	16	28	5
10	7	11	7	25	4
11	9	15	18	42	7
12	7	14	10	31	5
13	13	9	4	26	4
14	4	3	8	15	3
15	13	10	2	25	4
16	20	9	4	33	6
17	11	5	16	32	6
18	6	11	7	24	4
19	6	6	12	24	4
190	193	177	205	575	100

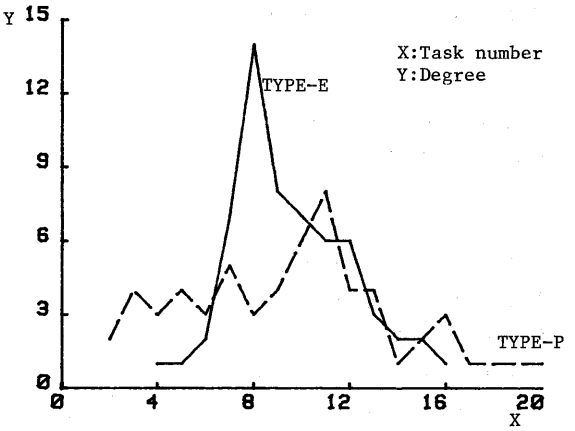

図6－3　度数分布図（ＴＹＰＥ－ＰとＴＹＰＥ－Ｅ）

表6－3　ＡＬＵの展開時のタスクの蓄積量（ＴＹＰＥ－Ｅ）
平均　９．５８　　分散　６．４１

Node	Port 0	Port 1	Port 2	SUM	%
0	16	10	12	38	7
1	7	9	15	31	5
2	12	9	15	36	6
3	8	7	13	28	5
4	11	6	13	30	5
5	11	10	13	34	6
6	11	8	10	29	5
7	8	10	10	28	5
8	8	8	11	27	4
9	5	8	8	21	4
10	9	12	14	35	6
11	9	10	7	26	5
12	12	11	9	32	5
13	7	9	4	20	3
14	8	7	14	29	5
15	15	8	7	30	5
16	8	8	6	22	4
17	10	9	12	31	5
18	9	11	8	28	5
19	9	8	7	24	4
190	193	177	205	575	101

表6－4　ＡＬＵの展開時のタスクの蓄積量（ＴＹＰＥ－ＥＳ）
平均　５．２８　　分散　２．７６

Node	Port 0	Port 1	Port 2	SUM	%
0	3	5	7	15	5
1	8	3	8	19	6
2	5	4	7	16	5
3	5	5	7	17	5
4	10	6	6	22	7
5	6	3	4	13	4
6	4	6	4	14	4
7	7	5	4	16	5
8	4	6	4	14	4
9	5	5	5	15	5
10	6	5	4	15	5
11	5	3	4	12	4
12	5	5	8	18	6
13	5	5	10	20	6
14	5	4	7	16	5
15	6	6	4	16	5
16	5	5	8	18	6
17	6	6	3	15	5
18	6	6	4	16	5
19	9	4	6	19	6
190	111	98	108	317	100

なっていることがわかる。

以上、３種類の展開法によるシミュレーション結果をもとに明らかになったことをまとめると、次のようになる。

（１）ＡＯＮ上には、タスクがほぼ均等になるように、展開させる必要がある。

（２）論理シミュレーションのデータが多い方が処理速度が速くなる。（パイプライン処理に近くなっている。）

（３）一つの木が他の部分木となる場合は、その部分木となる木を省略することができる。

（４）処理速度は、ＴＹＰＥ－ＥＳの場合、一台のコンピュータによる逐次処理に比し、約５．２６倍の処理速度を有する。

（５）シミュレーション実行時には、タスクのデータに待ち行列が生じる。これは、データフローによる処理を行なうため、回路中の遅延がそのままデータの処理待ちとなるためである。しかし、ＡＬＵのシミュレーションの場合にはタスクの最大待ち行列は２であり、シミュレーションの実行には、全く支障がない。

図６－４は、シミュレーション時のＣＰＵのタスク処理情況を描いたものである。処理されたタスクの量が各ＴＮＣ毎ほぼ一定の状態で処理されていることから、データが次から次へと流れ、処理が滞ることなく並列に進み、最終段階において急速にタスクの処理が減少して終了する様子がよくわかる。

７．　ＭＯＳ回路の論理シミュレーション

ＡＯＮによる論理シミュレーションの応用として、ＭＯＳ回路の論理シミュレーションについて考察する。ＭＯＳ回路の論理シミュレーションを行なう場合、ＭＯＳ回路を、論理素子の回路に等価変換して行なう場合がある。この方法を用いるならば、ＡＯＮによる論理シミュレーションは前述の方法で行なえる。しかし、ＡＮＤやＯＲのような回路は、直ちに論理素子に変換できるが、図７－１の

表6－5　シミュレーション結果

	Data	Calcurate	1-inst	TYPE	Queue
1	1	70	70	P	0
2	3	161	53.7	P	0
3	32	1264	39.5	P	0
4	128	5038	39.4	P	0
5	1024	39982	39	P	0
6	2048	79918	39	P	0
7	16384	639022	39	P	0
8	1	57	57	E	0
9	3	116	38.7	E	0
10	32	1015	31.7	E	0
11	128	3991	31.2	E	0
12	1024	31767	31	E	0
13	2048	63511	31	E	0
14	16384	507927	31	E	0
15	16384	311316	19	ES	0

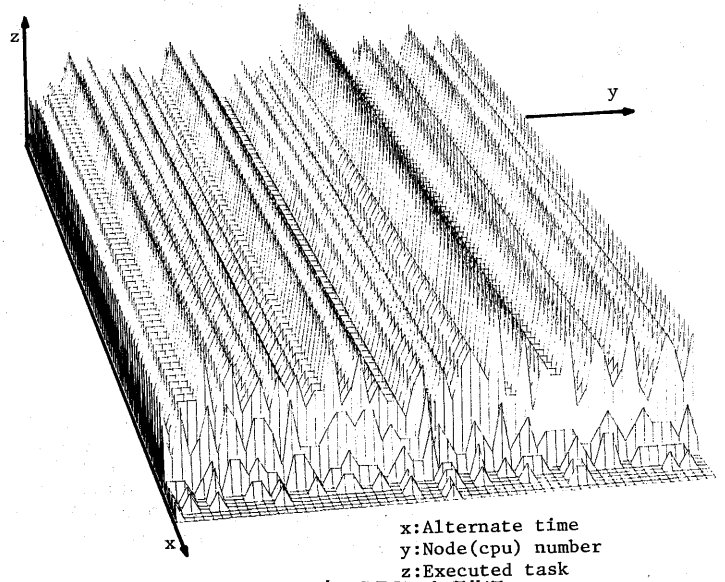

図6－4　ＡＬＵシミュレーション時のＣＰＵの処理状況

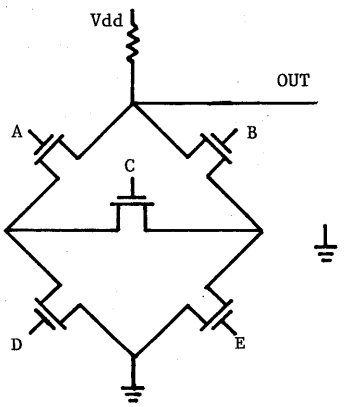

図7－1　ＭＯＳ回路例

ような回路では論理素子の回路に変換することは困難である。又、MOS回路を論理素子の回路に変換するために、その回路の関数機能を正確に変換できない恐れもある。MOS回路はそのままMOS回路として論理シミュレーションを行なう方がシミュレーション本来の意味からすると好ましい。　MOS回路の論理シミュレーションを行なう場合、MOSを単なるスイッチ動作をする素子と考えると、前述の回路との違いは、論理素子の回路では情報の流れが一方向に決まっているが、MOS回路では情報が双方向に流れるということである。このため、MOS回路の論理シミュレーションでは、前述の方法のように、データフローで実行することはできなくなる。そこで、MOS回路を論理シミュレーションする方法について、図7－1の回路を図2－2のAON上でシミュレーションする場合を例にして説明する。

図7－1の回路は、図7－2のように3ポートのMOS回路に変換することができ、これによりAON上に展開することができる。これをシミュレーションするためには、AONを半数づつ交互に動作させ、TNCにはその動作時に蓄積されたタスクすべてを処理させるという方法を用いる。AONが交互動作を行なう回数は、MOS回路を3ポートのMOS回路に変換した場合の回路で考える必要がある。図7－2の場合であれば、最大11回の交互動作を行なうことにより結果が出力される。

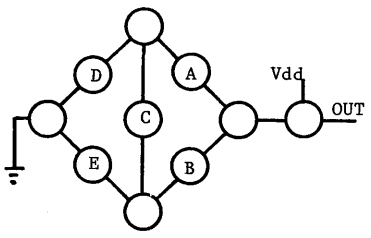

図7－2　MOS回路の3ポート表現

この最大回数は、出力端子からVddまたは、アース等の電源供給源となる端子までの最大経路の距離に相当する。ここに示した方法と、前述のシミュレーションとを比較すると次の点が異なる。

（1）タスクの演算子の種類は、MOSの動作すなわち開と閉を行なう演算子、および線の結合を行なう演算子の2種類で与えられる。

（2）前述の方法ではデータを順次流したので高速化できる効果（パイプライン効果）があった。MOS回路のシミュレーションでは、一つのデータのシミュレーションが終るまで次のデータは扱えないのでこのような効果はない。

MOS回路についても、AONによる論理シミュレーションができることが明らかになった。

8．　まとめ

本報告では、有限個のノードで構成される3ポートコンピュータネットワークによる並列論理シミュレーションについて新たな提案を行なった。この方式を用いることにより有限個のTNCをもつAONにおいて、論理素子の多い回路をネットワーク上に折りたたむように展開することによって並列論理シミュレーションが行なえることが明らかになった。また、本方式では、ネットワークの形状、展開を開始するノード等によって処理速度が異なるので、均等に展開できるようなネットワークを見出すことが今後の課題である。MOS回路のシミュレーションについては、AONによる論理シミュレーションの応用として述べたが本方式を用いることにより、MOS回路を直接シミュレーションすることが可能であることを示した。

参考文献

(1)M.Lacroix,A.Pirotte:DATA STRUCTURES FOR CAD OBJECT DESCRIPTION,18th Design Automation Conference,Paper 33.2,pp.635-659(1981)
(2)Gregory F.Pfister:THE YORKTOWN SIMUTATION ENGINE:INTRODUCTION,19th Design Automation Conference,Paper 7.1,pp.51-54(1982)