計算機アーキテクチャ59−10
オペレーティング・システム29−10
(1985・10・25)

# 二進木計算機向き並列計算プログラム

Parallel Computing Programs for the Binary-Tree Machine

高橋義造　　吉谷文徳

Yoshizo TAKAHASHI　　Fuminori YOSHITANI

徳島大学工学部情報工学科

Department of Information Science, Tokushima University

## 1。はじめに

単一の問題を分割して多数のプロセッサに分配し同時に計算を行なうことによって、短時間に解こうと言うのが並列計算であるが、その処理効率は使用する並列計算プログラムによってかなりの変動がある。効率の良い並列計算プログラムは、処理しようとする問題と使用する並列計算機の方式に適合するように作られなければならない。我々は二進木構造を持つ並列計算機に興味を持ち、１５台のプロセッサからなる二進木計算機ＣＯＲＡＬ’８３を開発し[1]、各種の問題の並列処理に使用してきた。二進木計算機はその構造から分割統治(divide-and-conquer)処理と、パイプライン処理に適応することが明かである。ＣＯＲＡＬ’８３を用いて開発された各種の二進木計算機向き並列計算プログラムについて、その方式と実測された処理効率について報告する。

図１にＣＯＲＡＬ’８３の構成を、また表１にその性能を示す。図１で、根にあるプロセッサをルートプロセッサ、一番下のレベルのものをリーフプロセッサ、その他のものをノードプロセッサと名付ける。またノードプロセッサの３個の接続方向をそれぞれ、上、左、右、またはTop, Left, Rightと呼んでいる。

並列計算プログラムの性能を評価するのに速度向上率（speed-up ratio）s(n)またはプロセッサ利用率(processor utilization) u(n)を用いることが通常に行なわれている。即ち　Ｔ１を１台のプロセッサによって計算した時の時間、Ｔｎをｎ台のプロセッサによって計算した時の時間とすると、

$$s(n) = T_1/T_n \qquad (1)$$

$$u(n) = s/n \qquad (2)$$

となる。

本論文では開発した並列計算プログラムを分割

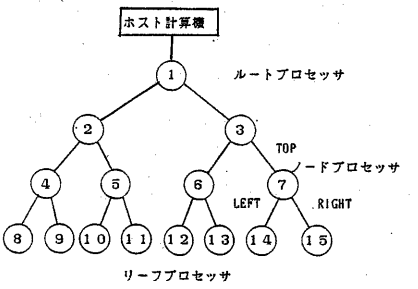



図１　ＣＯＲＡＬ’８３の構成図

表１　ＣＯＲＡＬ’８３の諸元

| 構成要素 | 項目 | 仕様 |
|---|---|---|
| ホスト計算機 | 機種 | ｉｆ８００／３０ |
| | ＣＰＵ | Ｚ８０Ｂ（５ＭＨｚ） |
| | メモリー | １２８ｋＢ |
| | ＯＳ | ＣＰ／Ｍ−８０ |
| 二進木プロセッサ | プロセッサ数 | １５台 |
| | ＣＰＵ | ｉ８０８５（２．５ＭＨｚ） |
| | ＲＯＭ | ８ｋＢ |
| | ＲＡＭ | １７ｋＢ |
| 計算機間通信 | ホスト・ルート間 | 直列転送９８００ビット／秒 |
| | プロセッサ間 | ８ビット並列転送９ｋＢ／秒<br>但しＣの関数では２ｋＢ／秒<br>割り込みでは　１ｋＢ／秒 |

統治型とパイプライン処理型に分類して夫々の方式と、ＣＯＲＡＬ’８３によって実行した結果得られた性能について説明し、さらにまた要求駆動型パイプライン処理とよぶ新しい方式を提案し、評価結果を示すこととする。

## ２。分割統治型並列計算プログラム

### ２。１　ツリーソート

ツリーソートの原理は図２に示す通りである。[2] 即ちソートするデータを等分し、ホストからすべてのプロセッサに分配し、各プロセッサがそれらを独立にソートしたのち、左右から送られるデータとマージして上に送る動作を行なう。

いまＮ個の要素をソートするものとして、ホスト・ルート間のデータ転送時間をｈ秒／要素、プロセッサ間のデータ転送時間をｐ秒／要素、１要素当たりのマージ時間をｍ秒とすると、プロセッサが１台の場合の計算時間Ｔ１は、

$$T_1 = 2Nh + S(N) \qquad (3)$$

と表わせる。またｎ台の場合の時間Ｔｎは図３のタイムチャートを参照することによって、次のように求められる。

$$T_n = 2Nh + 3Np[n-\log_2(n+1)]/n + Nm[2n-1-\log_2(n+1)]/n + S(N/n) \qquad (4)$$

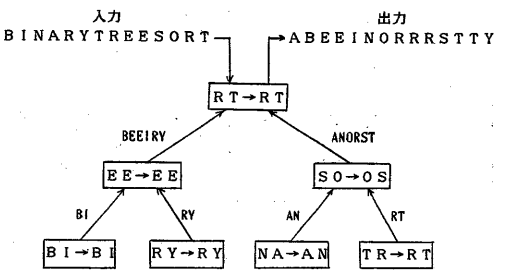



図２　ツリーソートの原理

遅いソートアルゴリズムを使うほど速度向上率は良くなるが、我々のプログラムではＳｈｅｌｌソートを用いた。これの計算時間は $N(\log_2 N)^2$ のオーダーであるから、データ転送時間を無視すれば速度向上率は

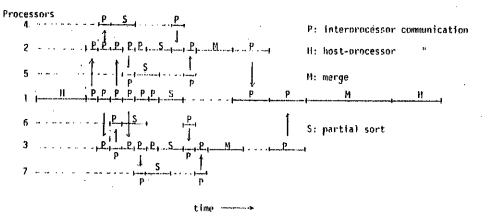



図３　タイムチャート

$$s(n) \simeq n/(1 - \log_2 n/\log_2 N)^2 \qquad (5)$$

となるはずである。プログラムの実行結果は表２のようになる。この結果に示すように１５台の場合に６倍の速度向上率が得られたが、ツリーソートのプログラムでは部分ソートは完全に並列に行なわれるが、マージの並列度が上のレベルのプロセッサになるほど減少するために効率はそれほど良くならないことがわかる。

表２　ツリーソートの実行結果

| n | data size | T(1) | T(n) | s(n) | u(n) |
|---|---|---|---|---|---|
| 3 | 3 kB | 35 sec | 20 sec | 1.9 | 0.63 |
| 7 | 7 " | 95 " | 36 " | 3.6 | 0.51 |
| 15 | 15 " | 295 " | 49 " | 6.0 | 0.40 |

### ２。２　Ｎクイーン問題

Ｎクイーン問題[3] は探索木を、バックトラックを繰り返しながら探索して解を探す問題であるが、ここでは探索木を多数ののサブツリーに分割し、これをプロセッサに等分に分配し、夫々が独立に解を探すようなプログラムを作った。一つの解が見つかると上のプロセッサに送るようになっている。プロセッサごとに割り当てられるサブツリーの数が同じでも、得られる解の数は異なるから負荷が等分されたことにはならない。従って１００％のプロセッサ利用率は期待できないが、図４の実験結果に示すように１５台で１０．９倍と、かなりの速度向上率が得られている。

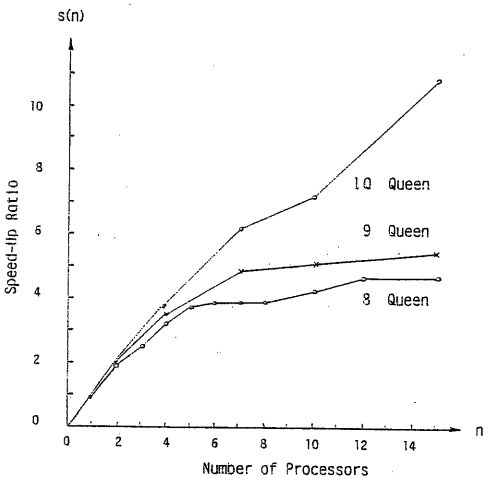



図４　Ｎクイーン問題の速度向上率

## ３。パイプライン処理型プログラム

### ３。１　素数計算

Eratosthenesのふるいによる素数計算[4] はパイプライン処理向きのアルゴリズムとして知られている。これは縦続に接続したプロセッサに自然数の列を入力し、これを各プロセッサが異なるふるいを用いて倍数であるかどうかを調べ、そうでないものだけを次のステージに送ることによって最終段から素数の列が取り出されると言う原理にもとずくものである。

このアルゴリズムをＣＯＲＡＬ’８３の二進木パイプライン向きに、次のように修正した。即ち求めるべき素数の上限をＭとすると、使用するふるいは３から $\sqrt{M}$ までの奇数であるので、これらのふるいの列をＬ個のグループに分ける。但しＬは二進木のレベル数である。そして下のレベルのプロセッサから小さいふるいのグループをわりあて、同じレベルのプロセッサが同じふるいを持つようにする。リーフプロセッサは異なる自然数の列を発生し、これを自分のふるいでテストし、倍数でないものだけを上のプロセッサに送る。このプロセッサは左右から送られる数のうち、小さい方を取って自分のふるいで調べ、倍数でないものだけをさらに上に送る。このようにしてホストにソートされた素数の列が送りだされる。

この方法で問題になるのは、ふるいの列をどのように分割すればプロセッサの負荷を均等にすることが出来るかということである。　図５はＭ＝１００の場合の、夫々のふるいを通過する自然数の数を示したものであるが、ある程度から上のふるいではこの数はほぼ一定であるから、我々のプログラムでは下のプロセッサは上のプロセッサの２倍の数のふるいを持たせるようにした。即ちレベルｖのプロセッサに割り当てるふるいの最大値Ｈｖ、最小値Ｌｖを次の式によって決める。

$$H_v = \sqrt{M} - (2^{v-1} - 1)m$$

$$L_v = \sqrt{M} - (2^{v} - 1)m \qquad (6)$$

ただし

$$m = \sqrt{M}/n$$

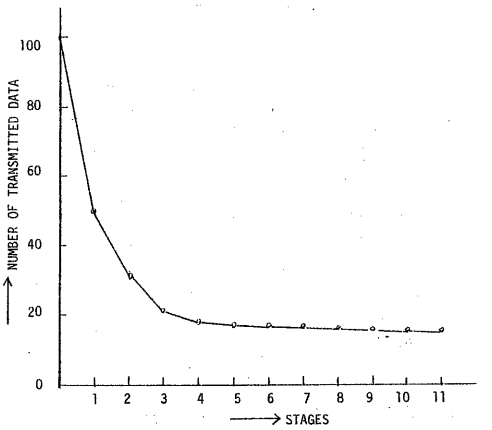



図５　ふるいを通過する自然数の数

表３にＭ＝40000 の場合の実行結果を示す。ホスト・ルート間の転送時間を除いたものでは１１．７倍という好結果が得られている。二進木パイプ

表３　素数計算の実行結果

| 項目 | 結果をホストに転送する場合 | 結果をホストに転送しない場合 |
|---|---|---|
| T(1) | ９４０秒 | ９２４秒 |
| T(15) | １８９〃 | ７９〃 |
| s(15) | ７．９ | １１．７ |
| u(15) | ０．５３ | ０．７８ |

ラインでは下のレベルのプロセッサほど高い並列度で動作するので効率が高いという特長がある。

### ３．２　ＦＦＴ

ＦＦＴは普通分割統治型のアルゴリズムと考えられているが、これを二進木パイプラインによって処理する方法を示す。二進木パイプラインは上からでも下からでも使用することができるが、ここではルートプロセッサから下方にデータを流すTop-downの方法によって説明する。

図６のデータフロー図に示す８点のＦＦＴについて考える。ここで○はバタフライ演算で、数字はその順序番号を示している。この計算は図７（ａ）のような４段の線形パイプラインによって同図（ｂ）のタイムチャートのように計算することができる。この場合、プロセッサ利用率は８×４÷１８÷４＝０．４４となる。

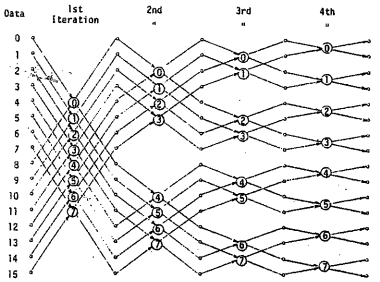



図６　ＦＦＴのデータフロー図

一般に$2^n$点のＦＦＴをｎ段の線形パイプラインによって処理する場合の計算時間Ｔは、ｔを１個のバタフライの計算時間とすると次のように計算される。

$$T_n = [(2^{n-1}+1)+(2^{n-2}+1)+\cdots +(2^1+1)+1]t = (2^n + n - 2)t \quad (7)$$

この計算にはｎ$2^{n-1}$個のバタフライ計算が含まれるから、プロセッサ利用率は次のようになる。

$$u(n) = 2^{n-1}/(2^n + n - 2) < 0.5 \quad (8)$$

即ち５０％以上のプロセッサ利用率を達成することはできない。

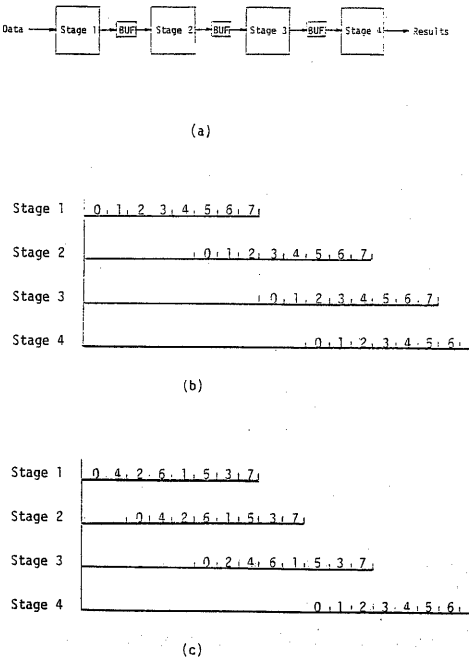



図７　線形パイプラインによる処理

つぎに　128 点のＦＦＴを３レベルの二進木パイプラインによって処理する方法を考える。７台のプロセッサの負荷が均等になるようにし、かつ各プロセッサが出来るだけ干渉しないようにするためには、図８のようにバタフライを分配すればよい。そしてルートプロセッサがビットリバース順にバタフライを計算すれば図９に示す順序でパイプライン処理が行なわれる。

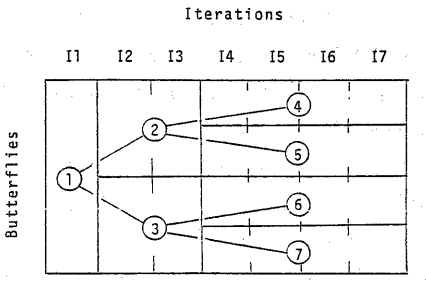



図８　バタフライの２進木への分配法

| PROCESSOR : | 1 | 2 | 2 | 4 | 4 | 4 | 4 |
|---|---|---|---|---|---|---|---|
| Iteration : | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| time | | | | | | | |
| 0 | 0 | · | · | · | · | · | · |
| 1 | 32 | · | · | · | · | · | · |
| 2 | 16 | 0 | · | · | · | · | · |
| 3 | 48 | · | · | · | · | · | · |
| 4 | 8 | 16 | · | · | · | · | · |
| 5 | 40 | · | 0 | · | · | · | · |
| 6 | 24 | · | 16 | · | · | · | · |
| 7 | 56 | 8 | · | · | · | · | · |
| 8 | 4 | 24 | · | · | · | · | · |
| 9 | 36 | · | 8 | · | · | · | · |
| 10 | 20 | · | 24 | 0 | · | · | · |
| 11 | 52 | 4 | · | 8 | · | · | · |
| 12 | 12 | 20 | · | · | · | · | · |
| 13 | 44 | · | 4 | · | · | · | · |
| 14 | 28 | · | 20 | · | · | · | · |
| 15 | 60 | 12 | · | · | · | · | · |
| 16 | 2 | 28 | · | · | · | · | · |
| 17 | 34 | · | 12 | · | · | · | · |
| 18 | 18 | · | 28 | 4 | · | · | · |
| 19 | 50 | 2 | · | · | 0 | · | · |
| 20 | 10 | 18 | · | · | 4 | · | · |
| 21 | 42 | · | 2 | 12 | · | · | · |
| 22 | 26 | · | 18 | · | 8 | · | · |
| 23 | 58 | 10 | · | · | 12 | · | · |
| 24 | 6 | 26 | · | · | · | · | · |
| 25 | 38 | · | 10 | · | · | · | · |
| 26 | 22 | · | 26 | 2 | · | · | · |
| 27 | 54 | 6 | · | 10 | · | · | · |
| 28 | 14 | 22 | · | · | · | · | · |
| 29 | 46 | · | 6 | · | · | · | · |
| 30 | 30 | · | 22 | · | · | · | · |
| 31 | 62 | 14 | · | · | · | · | · |
| 32 | 1 | 30 | · | · | · | · | · |
| 33 | 33 | · | 14 | · | · | · | · |
| 34 | 17 | · | 30 | 6 | · | · | · |
| 35 | 49 | 1 | · | · | 2 | · | · |
| 36 | 9 | 17 | · | · | · | 0 | · |
| 37 | 41 | · | 1 | · | · | 2 | · |
| 38 | 25 | · | 17 | · | 6 | · | · |
| 39 | 57 | 9 | · | · | · | 4 | · |
| 40 | 5 | 25 | · | · | · | 6 | · |
| 41 | 37 | · | 9 | 14 | · | · | · |
| 42 | 21 | · | 25 | · | 10 | · | · |
| 43 | 53 | · | 5 | · | · | 8 | · |
| 44 | 13 | 21 | · | · | · | 10 | · |
| 45 | 45 | · | 5 | · | 14 | · | · |
| 46 | 29 | · | 21 | · | · | 12 | · |
| 47 | 61 | 13 | · | · | · | 14 | · |
| 48 | 3 | 29 | · | 1 | · | · | · |
| 49 | 35 | · | 13 | 9 | · | · | · |
| 50 | 19 | · | 29 | 5 | · | · | · |
| 51 | 51 | 3 | · | · | 1 | · | · |
| 52 | 11 | 19 | · | · | 5 | · | · |
| 53 | 43 | · | 3 | 13 | · | · | · |
| 54 | 27 | · | 19 | · | 9 | · | · |
| 55 | 59 | 11 | · | · | 13 | · | · |
| 56 | 7 | 27 | · | · | · | · | · |
| 57 | 39 | · | 11 | · | · | · | · |
| 58 | 23 | · | 27 | 3 | · | · | · |
| 59 | 55 | 7 | · | 11 | · | · | · |
| 60 | 15 | 23 | · | · | · | · | · |
| 61 | 17 | · | 7 | · | · | · | · |
| 62 | 31 | · | 23 | · | · | · | · |
| 63 | 63 | 15 | · | · | · | · | · |
| 64 | · | 31 | · | · | · | · | · |
| 65 | · | · | 15 | · | · | · | · |
| 66 | · | · | 31 | 7 | · | · | · |
| 67 | · | · | · | · | 3 | · | · |
| 68 | · | · | · | · | · | 1 | · |
| 69 | · | · | · | · | · | · | 0 |
| 70 | · | · | · | · | · | · | 1 |
| 71 | · | · | · | · | · | 3 | · |
| 72 | · | · | · | · | · | · | 2 |
| 73 | · | · | · | · | · | · | 3 |
| 74 | · | · | · | · | 7 | · | · |
| 75 | · | · | · | · | · | 5 | · |
| 76 | · | · | · | · | · | · | 4 |
| 77 | · | · | · | · | · | · | 5 |
| 78 | · | · | · | · | · | 7 | · |
| 79 | · | · | · | · | · | · | 6 |
| 80 | · | · | · | · | · | · | 7 |
| 81 | · | · | · | 15 | · | · | · |
| 82 | · | · | · | · | 11 | · | · |
| 83 | · | · | · | · | · | 9 | · |
| 84 | · | · | · | · | · | · | 8 |
| 85 | · | · | · | · | · | · | 9 |
| 86 | · | · | · | · | · | 11 | · |
| 87 | · | · | · | · | · | · | 10 |
| 88 | · | · | · | · | · | · | 11 |
| 89 | · | · | · | · | 15 | · | · |
| 90 | · | · | · | · | · | 13 | · |
| 91 | · | · | · | · | · | · | 12 |
| 92 | · | · | · | · | · | · | 13 |
| 93 | · | · | · | · | · | 16 | · |
| 94 | · | · | · | · | · | · | 14 |
| 95 | · | · | · | · | · | · | 15 |
| 96 | · | · | · | · | · | · | · |

図９　二進木パイプラインの実行順序

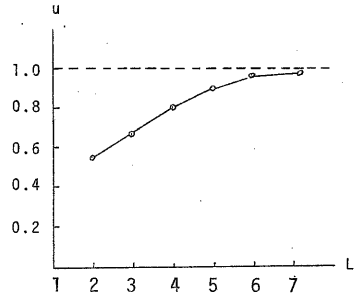



図１０　二進木パイプラインのプロセッサ利用率

図９からこの場合の実行時間は９６ｔであり、プロセッサ利用率は０．６７となって二進木パイプラインが線形パイプラインより優れていることがわかる。

一般に$n=2^L-1$　の場合に$2^n$ 点のＦＦＴをＬレベルの二進木パイプラインで処理したときの計算時間とプロセッサ利用率は次のように求められる。

$$T = 2^{2^L-2+1} + \sum_{v=1}^{L-1}(2^{2^{v+1}-v-1} - 2^{2^v-v-1}) \qquad (9)$$

$$u = 2^{2^L-2}/T$$

図１０に示すようにＬが大きくなるとｕは１に近づくことがわかる。

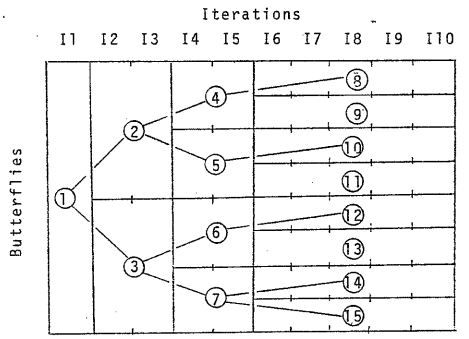



図１１　不均等な分配法

次に現実的な場合として上記のｎに対する条件が満たされない場合を考える。例えば１０２４点のＦＦＴを４レベルの二進木パイプラインで処理するものとする。このときには完全に負荷をバラ

ンスさせることは出来ないから図１１のような分配法を次善の策として取らねばならない。この場合にはシミュレーションによって次のような結果が得られる。

$$T_{15} = 640\ t$$
$$s(15) = 512\times10/T_{15} = 8.0 \quad (10)$$
$$u(15) = 0.53$$

このアルゴリズムに　従ってプログラムを作り、ＣＯＲＡＬ’８３で実行した結果は次の通りであった。

$$T_{15} = 29\ sec$$
$$s(15) = 6.9 \quad (11)$$
$$u(15) = 0.46$$

シミュレーションの結果ではプロセッサ間通信の時間を考慮していないために、実測値より良い結果がえられている。

## ４。　要求駆動型パイプライン処理

### ４。１　パイプライン処理に於ける負荷の均等化

効率の良い並列処理を行なうための重要な条件の一つは、負荷を全てのプロセッサに均等に配分することである。しかしながら、簡単な問題を除いては個々のプロセッサの計算量を事前に知ることは出来ないから予め負荷の均等配分を行なうことは不可能である。例えばある数が素数であるかどうかを検定する問題ではその数によって計算時間は大幅にかわる。そこで各プロセッサが一つの仕事を終われば次の仕事を要求する、いわゆる要求駆動(demand-driven) 方式を採用すれば、負荷が動的に均等配分され、全プロセッサが常に忙しく働くので効率の良い並列処理がおこなわれるのではないかと考えられる。

パイプライン処理では各段の計算時間が同一の場合に最大の効率がえられるが、このようにすることは実際には難しい。そこで後段のプロセッサが一つのデータの計算を終わると前段のプロセッサに次のデータを要求し、要求されたほうのプロセッサは実行中の計算を中断し、中間結果を後段におくり、自分は更に前段に新しいデータを要求する方式が考えられる。この方式を要求駆動型パイプライン処理と呼ぶことにする。図１２に通常のパイプラインと要求駆動型パイプラインの方式を比較して示す。通常のパイプライン処理は入力Ｘに対してｍ段のプロセッサで

$$Y = F_m \cdot F_{m-1} \cdot \cdots \cdot F_2 \cdot F_1 \cdot X \quad (12)$$

を計算するが、要求駆動型パイプライン処理ではｎ（＜ｍ）段のプロセッサで$F_1 \cdots F_m$ を分担して計算するものである。

また二進木パイプラインによる要求駆動型パイプライン処理も行なうことができる。

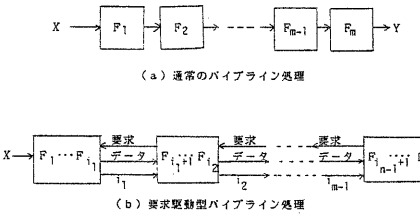



図１２　通常のパイプラインと要求駆動型パイプラインの比較

要求駆動型パイプライン処理が通常のパイプライン処理と違うところは次の点である。

１）パイプラインの各段が行なう計算は固定されておらず、その段への入力データと、後段より送られて来る要求信号によって決定される。

２）パイプラインの後段から前段に要求信号が送られる。

３）前段から後段に送られるデータには計算の中間結果の外に、どこまで計算が進んでいるかという情報が含まれる。

４）各段は全計算の中のどの部分でも実行することができる。

### ４。２　問題点と対策

要求駆動型パイプライン処理には負荷が自動的に均等配分されるという利点があるが、その一方で次のような問題点もある。

１）後段が要求信号を出してからデータを受け取るまでの間に待ち時間がある。

2）中間の段ですべての計算を終了してしまうことがあるが、この場合に後段が要求信号を送って来るまで待っているわけには行かない。

3）段間で転送するデータ量が多い。

これらの問題点への対策として

a）各段の中間に適当な容量のバッファをおくこと。

b）要求信号はできるだけ高い頻度で調べること。

c）計算の中断箇所はできるだけ中間データが少なくなるようにきめる。

などが必要である。

### 4。3　実行結果

CORAL'83を用いて線形パイプラインと二進木パイプラインによる要求駆動型パイプライン処理を行ない、その性能を実測した。先ず線形パイプラインによって8クイーン問題を計算した。その結果は表4に示すようになって、意外にも3段以上では殆ど効率は上がらないことがわかった。なおこの表および以下の表ではルートプロセッサからホストへの計算結果の転送は省略したものを示している。

次に二進木パイプラインによってNクイーン問題と素数計算を実行した。　その結果を表5と表6に示す。いずれの場合にも前章の、固定負荷配分方式に近い性能がえられるが、プロセッサ数が多くなるとやや劣るという傾向が観察される。

### 4。4　要求駆動型パイプライン処理のモデリング

線形パイプラインで要求駆動型パイプライン処理を行なうと、3段以上では性能が非常に落ちること、また二進木パイプラインの場合でもプロセッサ数が増えるとややこの傾向があることを解明するために、モデリングを行なった。先ず線形パイプラインの場合の初段のプロセッサの動作を状態遷移図で書くと図13（a）のようになる。ここでCは計算中、Tは要求信号のテスト、Sはデータ送出、Wは後段がデータを受け取れる状態になるまで待機している状態を表わす。Tは計算を中断する区切りのために入れたものであるが、この状態に止る時間は小さいので、近似的にこれを

表4　線形パイプラインによる8クイーン実行結果

| 段数 | 計算時間＊ | 速度向上率 |
|---|---|---|
| 1 | 49秒 | 1．0 |
| 2 | 33〃 | 1．48 |
| 3 | 31〃 | 1．58 |
| 4 | 30〃 | 1．63 |

＊結果をホストに転送しない場合

表5　二進木パイプラインによるNクイーン実行結果

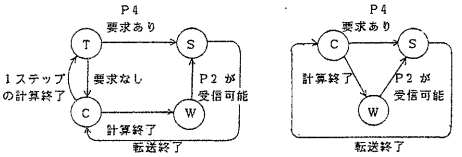
| 段数 | プロセッサ数 | N＝8 | | N＝9 | | N＝10 | |
|---|---|---|---|---|---|---|---|
| | | 時間 | s | 時間 | s | 時間 | s |
| 1 | 1 | 44秒 | 1 | 237 秒 | 1 | 1201秒 | 1 |
| 2 | 3 | 19 | 2.3 | 91 | 2.6 | 458 | 2.6 |
| 3 | 7 | 10 | 4.4 | 46 | 5.2 | 230 | 5.2 |
| 4 | 15 | 5 | 8.8 | 25 | 9.5 | 114 | 10.5 |

＊時間は結果をホストに転送しない場合のもの

表6　二進木パイプラインによる素数計算実行結果

| 段数 | プロセッサ数 | n＝1000 | | n＝10000 | | n＝40000 | |
|---|---|---|---|---|---|---|---|
| | | 時間 | s | 時間 | s | 時間 | s |
| 1 | 1 | 10秒 | 1 | 161 秒 | 1 | 947 秒 | 1 |
| 2 | 3 | 5 | 2.0 | 70 | 2.3 | 365 | 2.6 |
| 3 | 7 | 3 | 3.3 | 25 | 4.6 | 173 | 5.5 |
| 4 | 15 | 2 | 5.0 | 18 | 8.9 | 86 | 11.0 |

＊時間は結果をホストに転送しない場合のもの




（a）　　（b）

図13　要求駆動型パイプラインの初段の状態遷移

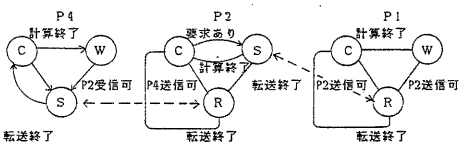



図14　要求駆動型パイプラインの各段の状態遷移

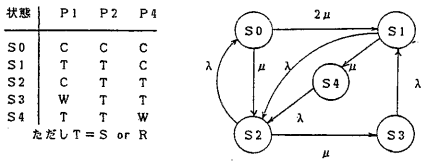



図１５　線形要求駆動型パイプラインの
システム状態遷移図

省略した（ｂ）の図を用いることにする。
従って３段の線形要求駆動型パイプラインの各プロセッサの状態遷移図は図１４のようになる。Ｒはデータ受け取りの状態である。ここで初段Ｐ４のＳと中段Ｐ２のＲ、中段のＳと終段のＰ１のＲはいずれも同時に存在することを考えると、このシステムには図１５に示すＳ０からＳ４までの５個の状態しか存在せず、それらの間の遷移図は同図に示すようになる。いますべてのプロセッサが計算を終了する確率をμ、転送が終了する確率をλとし、これらが指数分布をするものと仮定すれば各状態間の遷移確率は図に記入したようになる。状態Ｓ０では３台の、状態Ｓ１とＳ２では１台のプロセッサが稼働するので、このシステムの速度向上率は次のように計算される。

$$s(3) = 3p_0 + p_1 + p_2$$

$$= \frac{3 + 8\rho + 6\rho^2}{(1 + \rho)(1 + 2\rho)(1 + 3\rho)} \qquad (13)$$

ただし　$\rho = \mu / \lambda$とする。

同様にして２レベルの二進木プロセッサによる要求駆動型パイプラインの各プロセサ、およびシステムの状態遷移図はそれぞれ図１６、図１７のようになる。

これより速度向上率は次のように得られる。

$$s(3) = \frac{3 + 3\rho}{1 + 3\rho + 3\rho^2} \qquad (14)$$

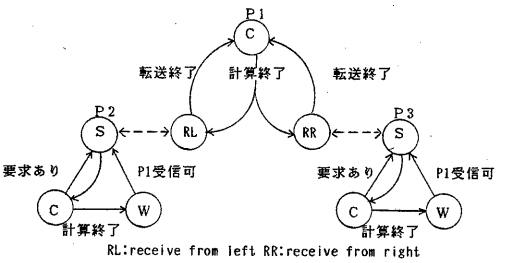



図１６　二進木要求駆動型パイプラインの
各段の状態遷移図

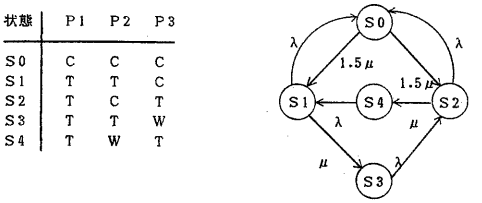



図１７　二進木要求駆動型パイプラインの
システム状態遷移図

## ５。結論

二進木計算機は構造が単純であるのにかかわらず、プロセッサ間距離と、ルートプロセッサからの放送時間が短いという特長を持つために、特に構造を持たない問題や、木構造以外の、例えば偏微分方程式など格子構造の問題にたいしても十分有効であるが、[5][6]本研究では特に二進木構造に分解できる問題を取り上げ、並列計算プログラムの構成法と性能の実測結果にもとずく評価を行なった。取り上げたプログラムは固定負荷配分を行なう分割統治型とパイプライン処理型、および自動的に均等負荷配分を行なう要求駆動型パイプライン処理型の３種類である。評価の結果はこれらの並列計算プログラムの方式が、今後の実用的な並列処理の応用分野で十分活用されることを期待させるものである。

## 参考文献


［１］吉谷、高橋、「２進木構造並列プロセッサＣＯＲＡＬプロトタイプ８３の性能評価」情報処理学会29回大会（昭５９）５Ｂ－１

［２］Ｍｅａｄ，Ｃ．，Ｃｏｎｗａｙ，Ｌ．，菅野他訳「超ＬＳＩシステム入門」培風館（昭５６）PP. 324-327

［３］例えば中西「Ｌｉｓｐ入門－システムとプログラミング」近代科学社（１９８１）PP.77-79

［４］例えば黒川「ＬＩＳＰ入門」培風館（１９８２）PP. 123-126

［５］Takahashi,Y.,Nobutomo,Y.,Kawamura,T.,” Strategies and Performance Evaluation of Parallel Computation in Solving the Laplace Equation”,Journ. Information Processing,Vol. 5,No.4,(1982)pp.239-246

［６］桑原、吉谷、高橋「二進木マシンによる偏微分方程式の並列計算方式」電子通信学会技術研究報告Vol.84,No.282 （昭６０）ＣＡＳ－１８６