卒業論文

非同期式並列処理の基礎的研究

東北大学工学部通信工学科
本田健

目 次

図目次

第 1 章

序論

1.1 本研究の背景と目的

現在、コンピュータの処理速度の向上が望まれている（1.2節）が、それを実現する方法の１つにシストリックアレーがある。

シストリックアレーは、H.T.Kung によって 1978 年に提案され同期的にデータを流す様子が心臓の収縮による血液の流れに似ていることからその名前が付けられた。シストリックアレーとは単純な機能を持ったプロセッサ（セルと呼ぶ）を規則的に並べ、局所的に結合したものである。各セルの機能は同じであるため、繰り返しのアルゴリズムを高速に処理することができ、配線が局所的であるため VLSI 化も容易であるなどの利点を持つ。このアレーにデータを入力することにより各セルで並列に計算が行なわれ、結果が出力される。しかし、このシストリックアレーは同一のクロック信号に同期して動作するため、すべてのセルに同一のクロック信号を入力しなければならない。このことは、アレーの大きさが大きくなると難しくなる。

そこで、非同期式のシストリックアレーが 1982 年に S.Y.Kung 等によって提案された。この非同期式シストリックアレーは、各セルがクロックをもち、データが揃ったら計算を行なうという動作をする。そのため、グローバルクロックを必要としない、などの利点がある。欠点としては、動作の確認が難しいことが挙げられる。

そこで、本研究では非同期式シストリックアレーのシミュレータ Spasa(Simulation and Projection for Asynchronous Systolic Algolism) を作成する。

シストリックアレーと非同期式シストリックアレーの比較は 2.3 節 で行なう。

1.2 コンピュータの処理速度

現在のコンピュータは一昔前とは比べものにならないほどの処理速度を持っている。それにもかかわらず、信号処理、画像処理、物理現象のシミュレーション等のさらなる処理速度の向上を望む分野も多い。

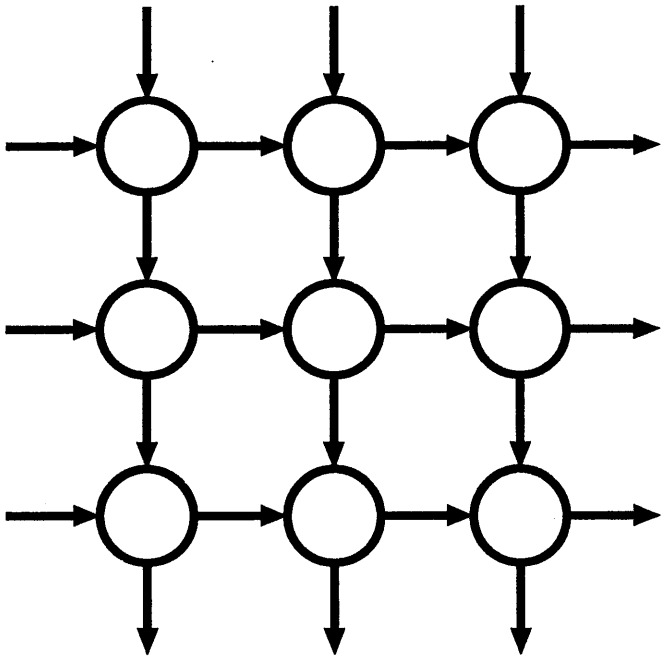

Figure 1.1: シストリックアレー

よって、コンピュータの処理速度の向上は重要な課題であるわけだが、現在考案、研究されている方法には大きく分けて次のような手段がある。

- アーキテクチャによる処理速度の向上
- デバイスによる処理速度の向上

アーキテクチャによる処理速度の向上として考えられているものとしては、並列処理アーキテクチャによる実現がある。

並列処理アーキテクチャは大きく分けて次のようなものがある。

- パイプライン処理
- マルチプロセッサシステム
- アレープロセッサ

パイプライン処理は、1つの処理をいくつかのステージに分けて、それにデータを流し込んでゆく。パイプラインに続けて次次とデータが入った場合、各ステージは並列に動作する。

例えば、命令実行機構のパイプライン化してみる。命令実行の処理は、フェッチ、デコード、オペランドフェッチ、実行の4ステージに分けることができる。その各処理を同時に実

行できるように作れば、命令やオペランドを次次と入力することによって、スループット（単位時間に完了する命令数）を向上させることができる。今の場合、各ステージにかかる時間が同じだとすれば、（この仮定は現在の技術とてらし合わせれば妥当である。）命令数が多くなった場合、スループットはパイプライン化しない場合に比べて4倍になる。

Figure 1.2に3つの命令がパイプライン化されて実行される様子を示す。

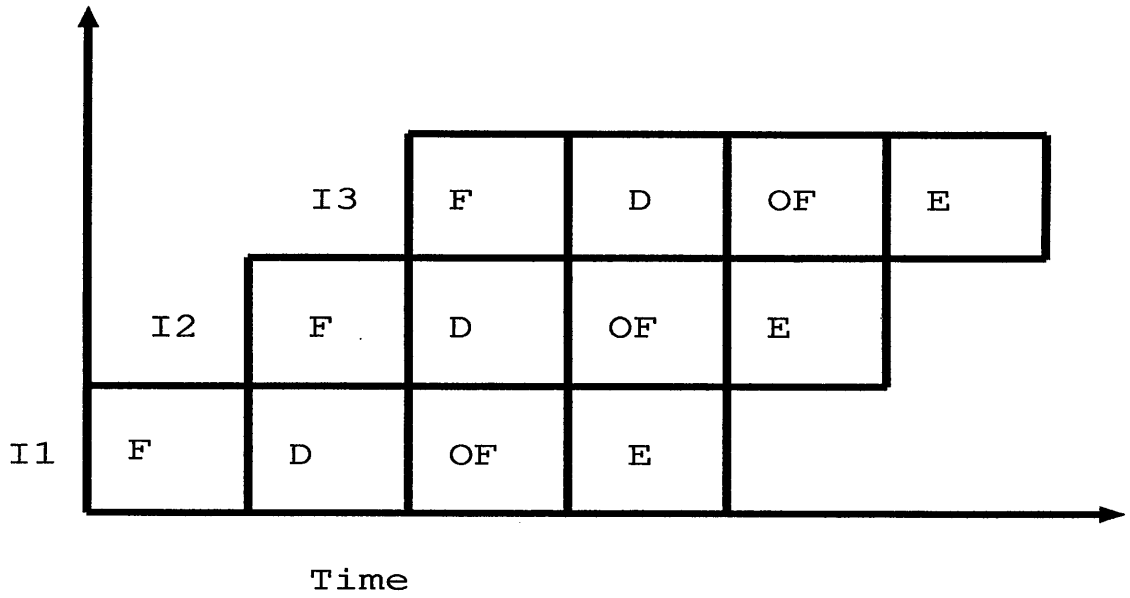

Figure 1.2: 命令実行機構のパイプライン

第 2 章

同期式シストリックアレーと非同期式シストリックアレーの比較

この章では、同期式と非同期式のシストリックアレーの比較をおこなう。

2.1 同期式シストリックアレー

同期式シストリックアレーは、Figure 2.1のようにセル間にラッチを設け、そのラッチに同一のクロック信号を流し、データを同期的に流す。セルの中身は計算をする機能だけで良く、単純なものになる。しかし、アレーが大きくなった場合に、同一のクロック信号を行き渡らせるのが難しくなってしまう。

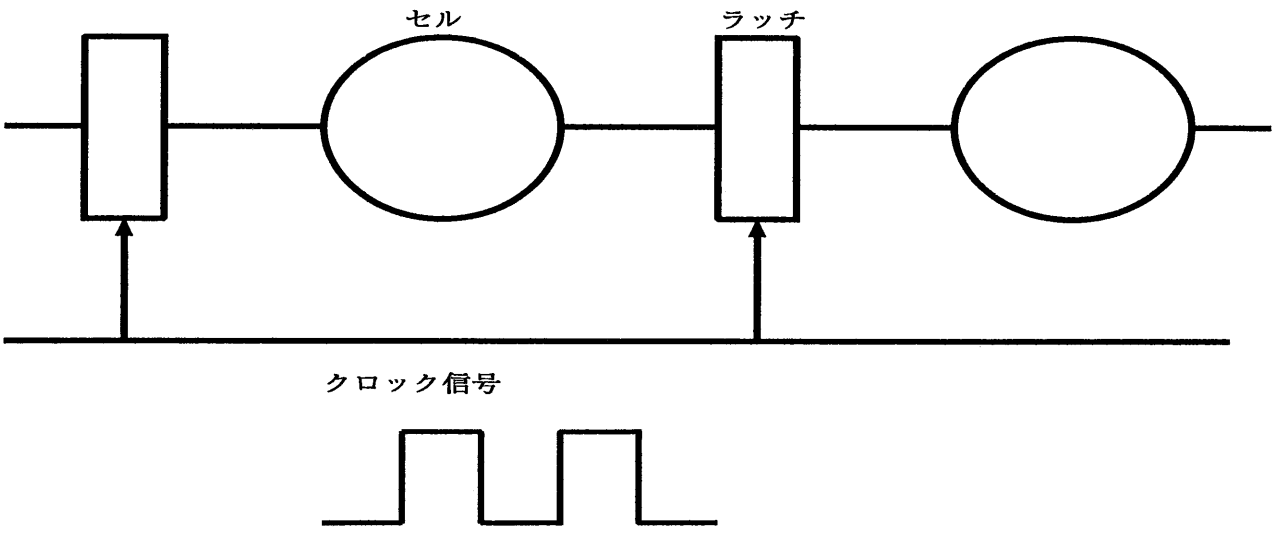

Figure 2.1: 同期式シストリックアレー

2.2 非同期式シストリックアレー

非同期式シストリックアレーは、Figure 2.2のように各セルがクロック回路を持ち、セル間には（FIFO）バッファがある。セルは入力バッファにデータが揃っていれば、計算を行ない、出力バッファ（あるセルにとっては入力バッファ）に結果のデータを出力する。この結果、グローバルクロックは必要なくなるが、セルの中にクロック回路やセルの挙動を書いておくROMなどが必要になり、計算をする以外の回路が多くなってしまう。また、同期式に比べて動作の確認が難しくなる。計算時間の点からいえば、理論的には非同期式シストリックアレーの方が計算時間が短くなる場合がある。(2.3節)

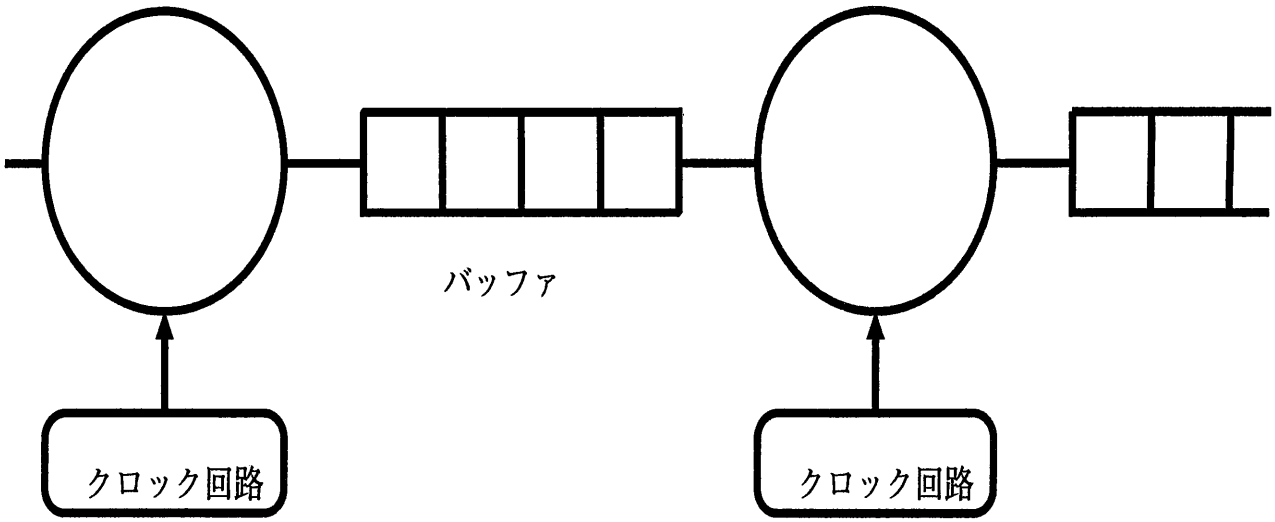

Figure 2.2: 非同期式シストリックアレー

2.3 同期式シストリックアレーと非同期式シストリックアレーの動作の比較

マッピングするアルゴリズムによっては、非同期式のほうが同期式よりも計算時間が短くなる場合がある。以下にその例を示す。

今、セルの構造を、横方向にはデータが入ってから結果のデータが出力されるまで2クロックかかり、縦方向には1クロックかかるものとする。このようなシストリックアレーに各入力部にデータを1個ずついれる。この場合の同期式シストリックアレーの動作はFigure 2.3のようになる。また、非同期式シストリックアレーの動作はFigure 2.4のようになる。(ただし、白いセルは計算が行われていなく、黒いセルは計算が行われていることをあらわし、Tは1クロックの時間を表す。)

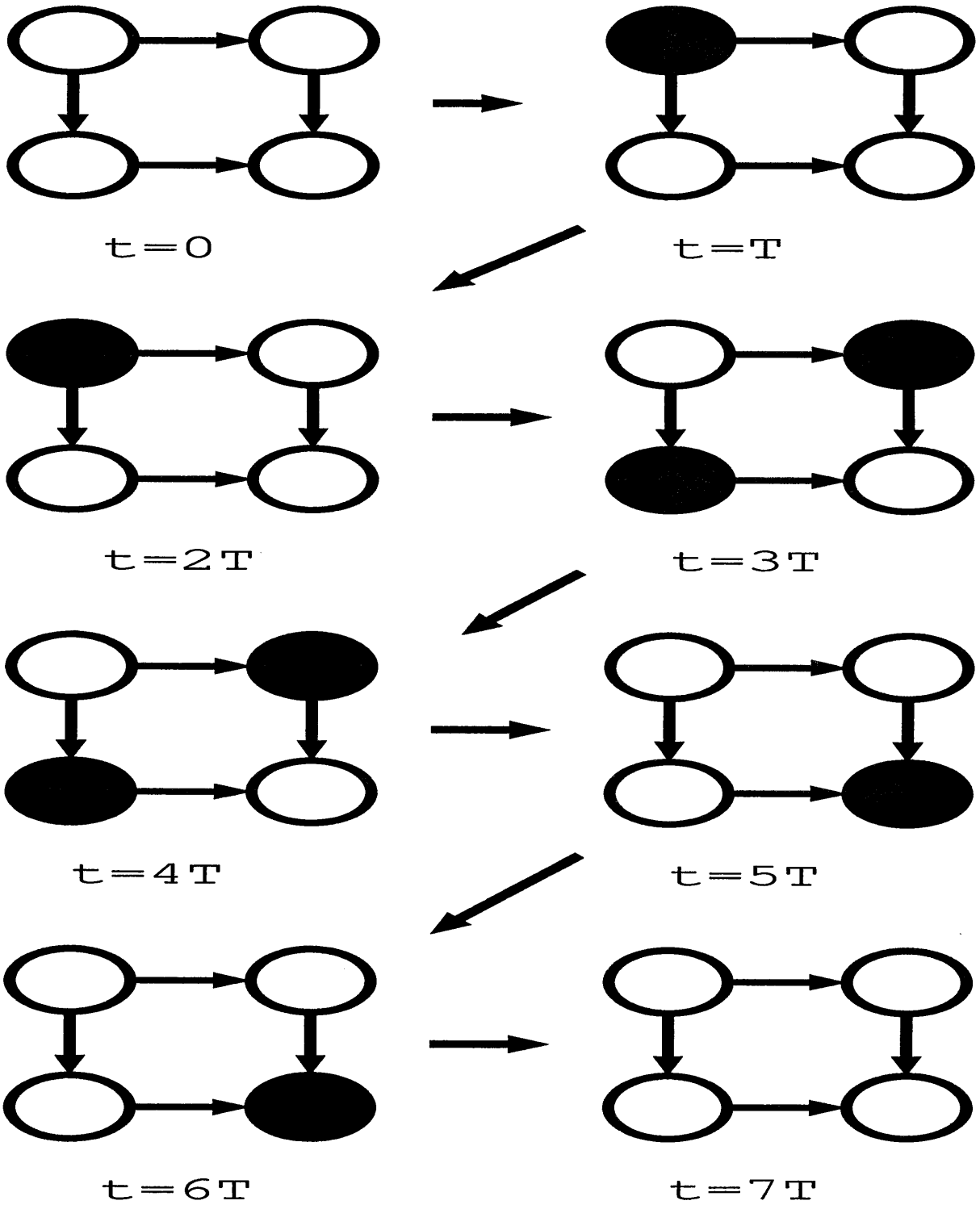

Figure 2.3: シストリックアレーの動作

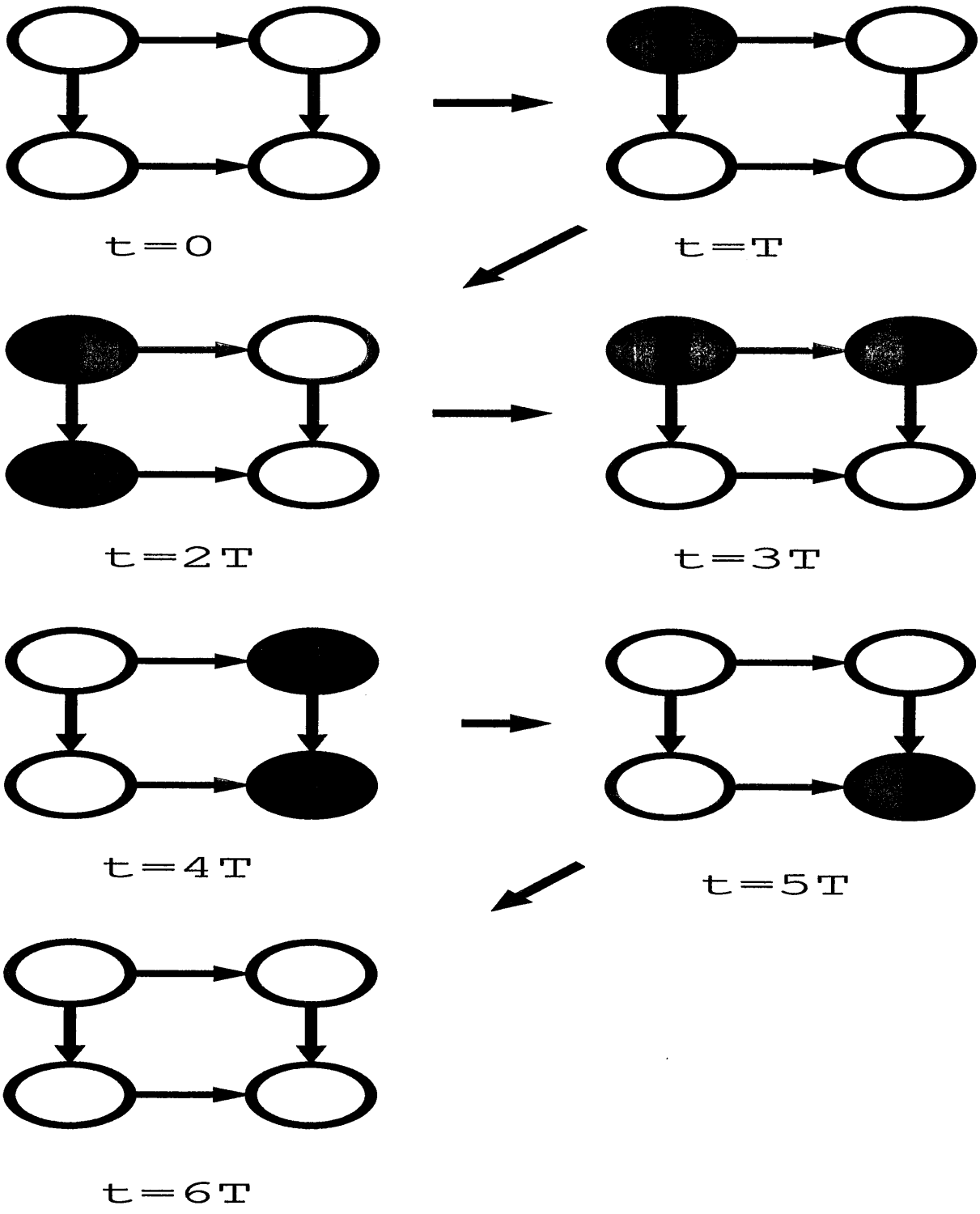

Figure 2.4: 非同期式シストリックアレーの動作

第 3 章

シストリックアレーの設計法

この章では Kung [2] のシストリックアレーの設計の例を見る。この方法によれば、与えられたアルゴリズムからシストリックアレーを導くには次の 3 ステップを踏む。

1. 与えられたアルゴリズム（プログラム）から、依存グラフ（Dependence graph) を導く。
2. 依存グラフをある方向に射影 (projection) して、シグナルフローグラフを作る。
3. シグナルフローグラフからシストリックアレーを導く。

以下に、1 から 3 の説明を述べる。

1. 依存グラフは、与えられた for ループのアルゴリズムに対して、1 つの繰り返し単位を 1 つのセルに割り当てて結合したものである。
2. シグナルフローグラフは、それをある方向に射影したものである。射影とは依存グラフ（一般的にはシストリックアレー）をある方向に押し潰して、同じ計算をするシグナルフローグラフ（シストリックアレー）を作ることである。(セルの機能は射影前と射影後では同じ)
3. 得られたシグナルフローグラフは、いわばシストリックのモデルなので、それから実際にシストリックアレーや、非同期式シストリックアレーを作ることである。

例として、ソーティングのアルゴリズムを実現するシストリックアレーについてその設計法を適応してみる。

ソーティングのアルゴリズムは以下のように書ける。

```
for(i=1;i<N;++i)
    for(j=i;j<N;++j)
       {
           m[i][j+1]=max(x[i][j],m[i][j]) ;
           x[i+1][j]=min(x[i][j],m[i][j]) ;
        }
```

1. このソーティングアルゴリズムにより、セルの機能は、for ループの中身、つまり入力された２つのデータの大きいデータを片方に、小さい方をもう片方に出力する、ものとなる。セルの構造は Figure 3.2のようになる。また、依存グラフは出力されたデータが使われる（入力される）繰り返し単位を考えることにより、Figure 3.1のようになる。

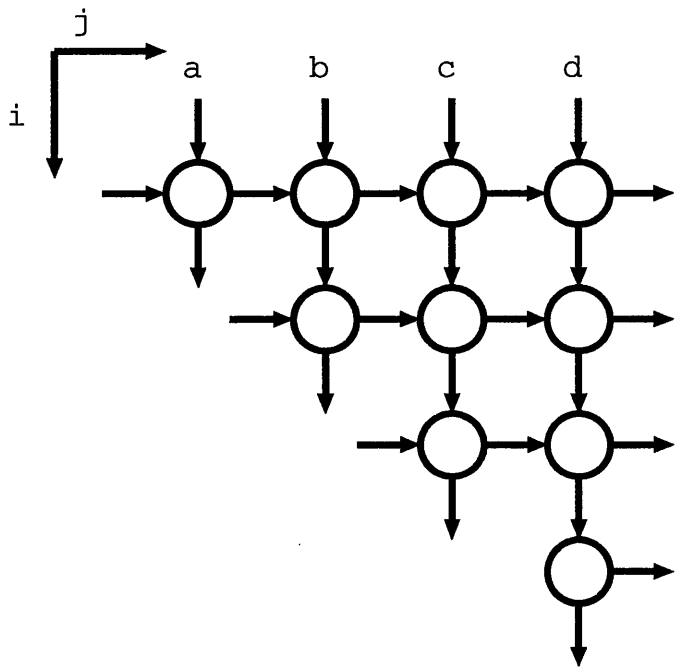

Figure 3.1: ソーティングアルゴリズムの依存グラフ

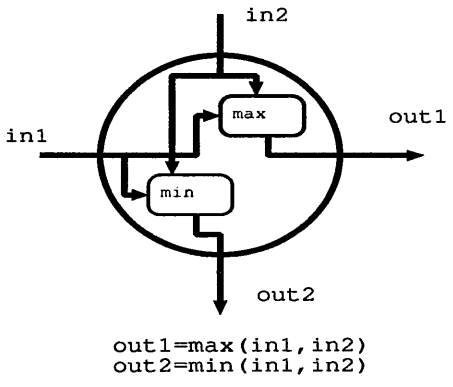

Figure 3.2: セルの構造

2. 得られた依存グラフをある方向に射影することにより、ソーティングを行なういろいろなシストリックアレーを導くことができる。Figure 3.3に、(i,j)=(0,1),(1,0),(1,1) の3方向に射影したシストリックアレーを示す。ただし、各シストリックアレーの脇についているベクトルはそのシストリックアレーがその方向に射影されたものであることを示し、a,b,c,d はソートされるデータ（入力データ）を表す。

3. 得られたシグナルフローグラフを実際に同期式や非同期式にマッピングする。

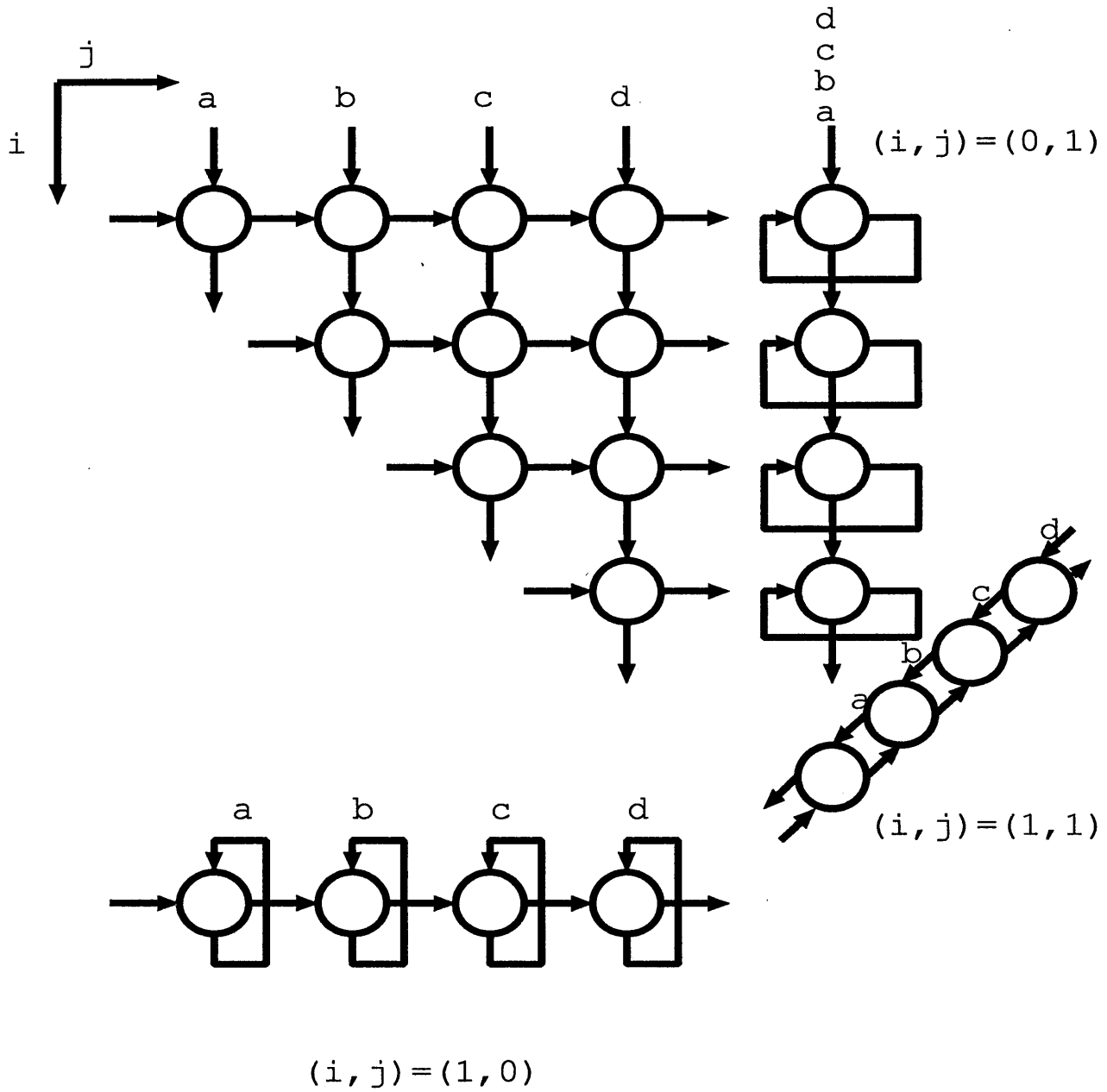

Figure 3.3: ソーティングアルゴリズムの依存グラフの射影

第 4 章

シミュレータ Spasa の説明

この章では作成した非同期式シストリックアレーのシミュレータ Spasa の説明をする。

4.1 シミュレートの流れ

Spasa は、シミュレートを行うプログラムと、射影を行うプログラムに分かれる。シミュレートを行うプログラムに、シストリックアレーの構造を記述した仕様ファイルを入力すると、そのシストリックアレーのシミュレートを行なう C++のソースファイルを出力する。それをコンパイルすることによりそのシストリックアレーのシミュレータができる。それに入力データを入力して実行することにより、計算結果とその結果のデータが出力される時刻を出力する。

また、定義したシストリックアレーに射影を行なわせたい場合は射影ベクトルをつけて射影するプログラムに入力すると、射影したシストリックアレーを定義した仕様ファイルを出力する。それをシミュレートをするプログラムに入力してすることによりシミュレートを行なうことができる。

Figure 4.1にシミュレートの流れ図を示す。

4.2 Spasa に入力するシストリックアレーの仕様ファイルの書式

ここでは、Spasa に入力するシストリックアレーの構造を記述する仕様ファイルの書式について述べる。

シストリックアレーの構造を決定するには、以下の 3 つを定義すればよい。

- セルの構造（機能）
- セル間の結線
- どこからデータを入力するか

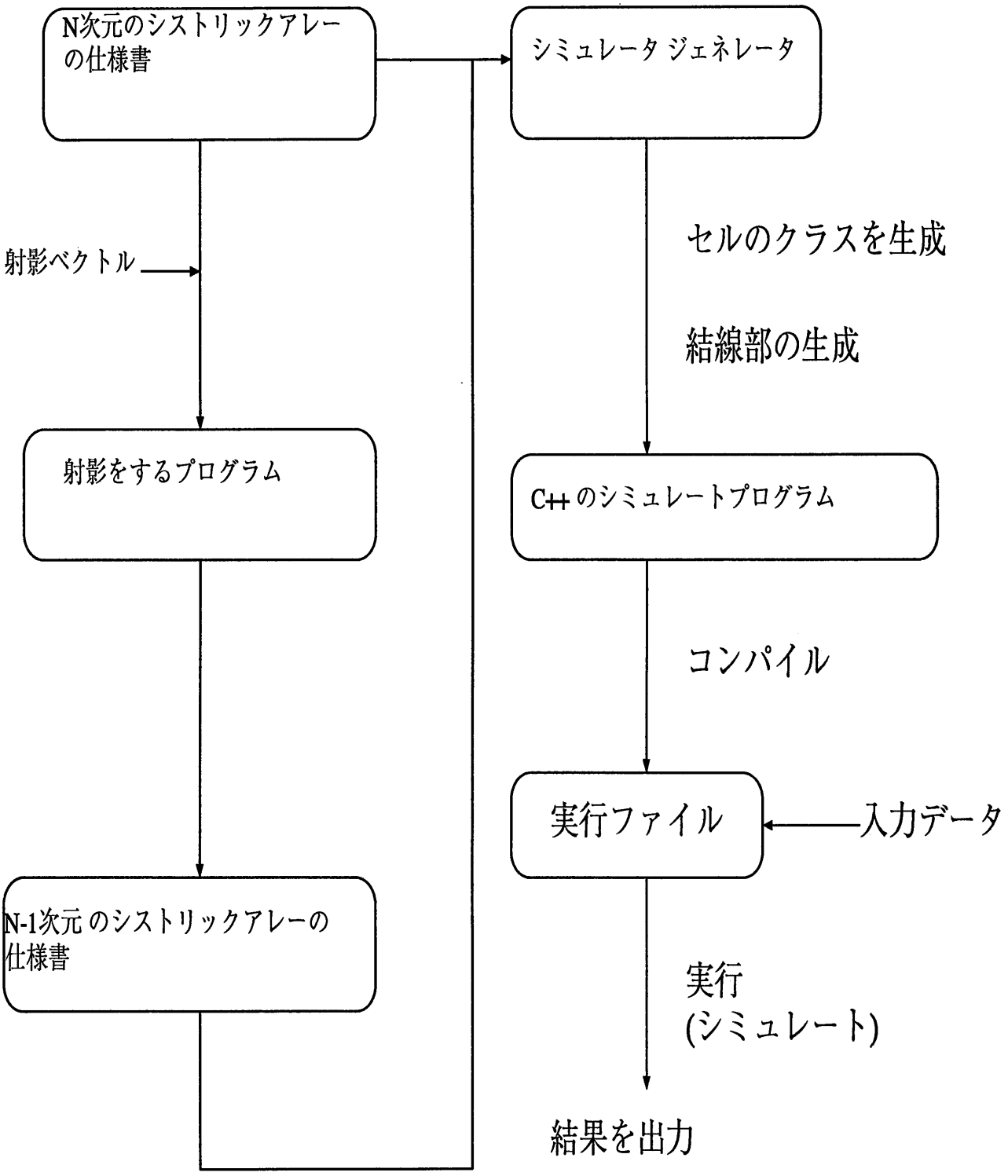

Figure 4.1: シミュレートの流れ

Spasa では、これらを以下のような文法で定義する。

```
cell{
```

セルの定義

```
};

array{
```

アレーの定義

```
};
```

以下にセルとアレーの定義の仕方を述べる。

4.2.1 セルの定義の仕方

セルの定義をするには

- 入力ポート
- 出力ポート
- セルの中でおこなわれる計算

を定義すればよい。まず、入力ポートと出力ポートであるがこれは以下のように記述する。

```
inport [入力ポート名] , [入力ポート名] ... ;
inport [入力ポート名] , [入力ポート名] ... ;
...
outport [出力ポート名] , [出力ポート名] ... ;
outport [出力ポート名] , [出力ポート名] ... ;
...
```

次に、セルの中でおこなわれる計算の記述であるが、これはセルが非同期動作をするということを強調して、

```
[入力ポート名].recv() ;
.....
[出力ポート名].send([式]) ;
.....
```

のように記述する。

4.2.2 アレーの定義の仕方

アレーを定義するには

- セルを何個、どのように用意するか
- どのセルのどの入力ポートからデータを入力するか
- セルをどのように結線するか

を定義すればよい。
セルの宣言は、配列のように宣言する。例えば、2次元で4x4の宣言をしたければ、cell[4][4]のようにする。
セルのポートを指定したい場合には、

[セル名].[ポート]
とする。

データを入力するポートは、頭にinportをつけて宣言する。また、シミュレートの結果を見るために、頭にoutportをつけて、アウトポートを宣言する。結線は、

[出力ポート]->[入力ポート]

のようにして、forループを用いて書く。
最後に、結果の見たいポートを宣言したアウトポートと結ぶ。

[結果をみたい出力ポート]->[宣言したアウトポート]

5.1節に定義例を示す。

4.3 シミュレートのアルゴリズム

4.3.1 シミュレートのアルゴリズム

SpasaではセルをFigure 4.2のようなデータ構造で実現している。Figure 4.2で、クラスDATAは、データの値とその生成された時刻をメンバとする。つまり、アレー中を流れるデータはその値だけでなく、そのデータが生成された時刻も記憶しているものとする。

シミュレートのアルゴリズムを説明する。まず、セルをキューに並べる。そして、シミュレートを開始する。まず、セルをキューから出して、そのセルのすべての入力キュー（入力ポート）にデータがあれば、そのセルは計算できる状態にあるので、計算（次節）を行ない、出力データを出力キュー(出力ポート)に出力する。計算が終ったら、キューに入れる。もし、すべての入力キューにデータがなければ、そのセルは計算できない状態にあるので、なにもしないでキューに戻す。もし、キューにあるすべてのセルの入力キューが空なら、シミュレート完了で、結果を見るように指定されたキュー（ポート）に入っているデータを表示して終了する。(そのデータは生成された時刻も含んでいるので、計算時間もわかる。）そうでなければ、またセルをキューから出して、シミュレートを行なう。Figure 4.3にシミュレートのアルゴリズムの流れ図を示す。

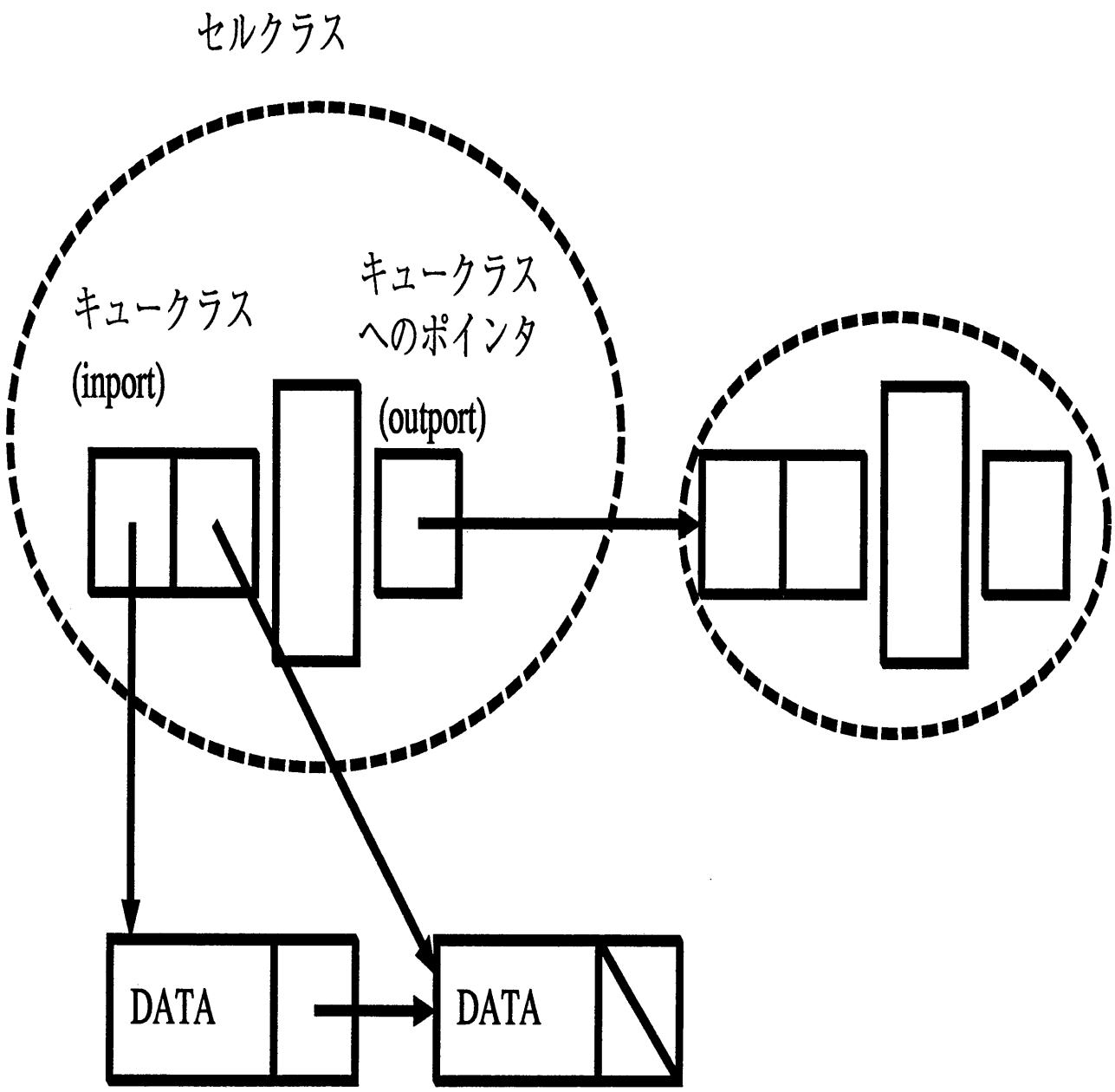

Figure 4.2: セルのデータ構造

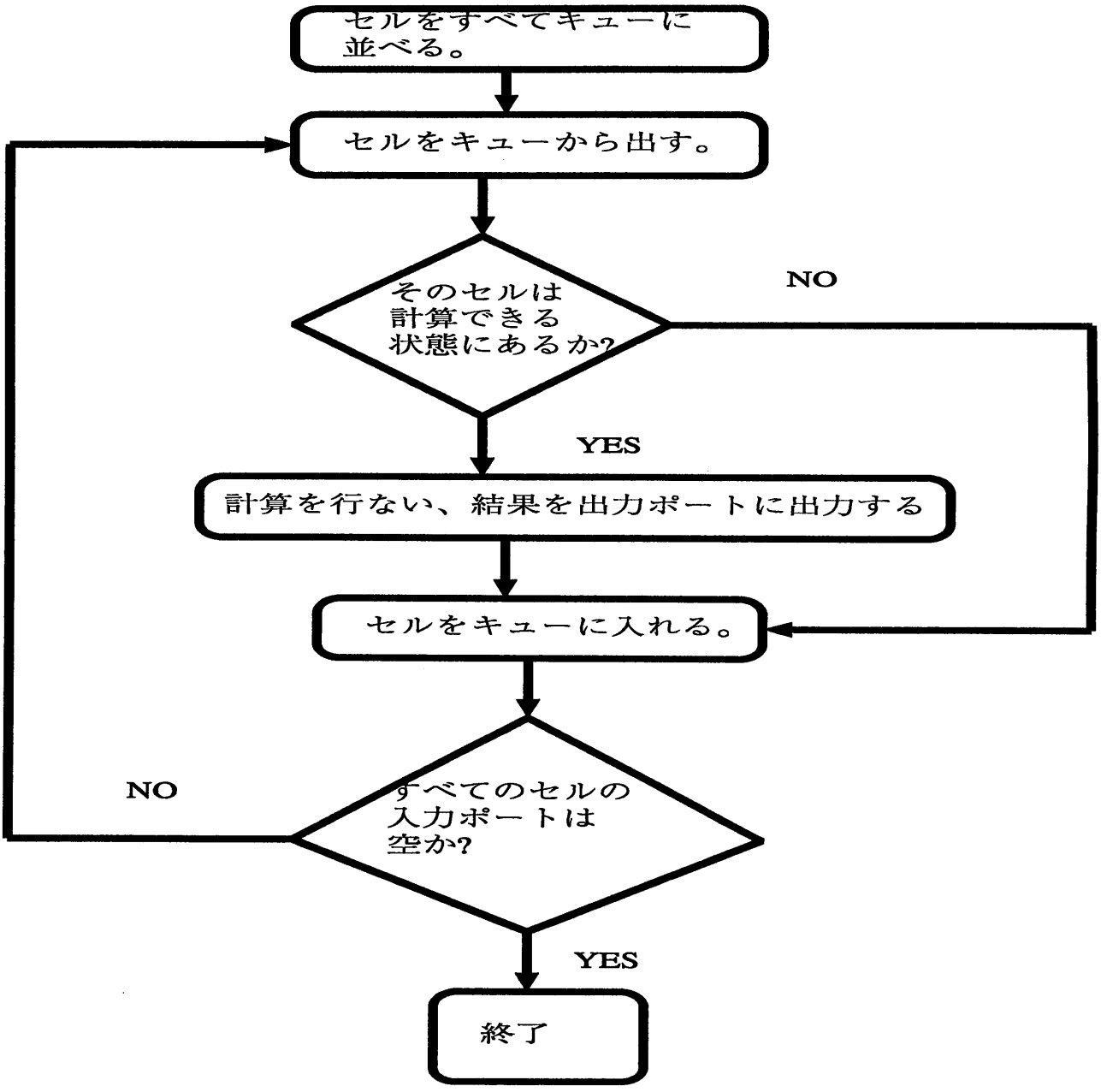

Figure 4.3: シミュレートのアルゴリズム

4.3.2 セルの中で行なわれる計算のアルゴリズム

セルで行なわれる計算のアルゴリズムについて説明する。
セルは時刻を記憶している。入力データが入ると、セルはすべての入力データの生成された時刻をみる。それらの時刻と、セルの記憶している時刻を比較する。

- セルの記憶している時刻のほうがすべての入力されたデータの生成された時刻よりも早い場合。
  セルが入力データが揃うのを待っていたということなので、セルの記憶している時刻をすべての入力データの生成された時刻の中で最大の時刻に合わせる。

- そうではない場合。
  すべての入力データが揃った後にセルが計算できる状態になったということなので、セルの記憶する時刻はそのままでよい。

そして計算を行ない、出力データを出力する。ただし、出力データの生成された時刻は
(セルの記憶している時刻)+(計算にかかった時刻)
とする。

4.4 Spasa のにおける射影ベクトルの制限

現在のところ、Spasa には射影をおこなうプログラムに入力する射影ベクトルには制限がある。それは以下のようである。

- 2 次元から 1 次元の射影は任意の方向

- 3 次元以上は基本ベクトルのみ

第 5 章

動作確認

この章では、作成した Spasa に仕様ファイルを入力し、動作確認をおこなう。

5.1 入力する仕様ファイル

動作確認のために入力する仕様ファイルとしてここでは Figure 5.1のような機能のシストリックアレーを定義した仕様ファイルを入力する。

```
(1,-1) ... 射影ベクトル

int i,j ;

cell   ... セルの構造の記述
{
    inport  a_in ;
    inport  b_in ;
    outport c_out ,d_out ;

    a_in.recv() ;
    b_in.recv() ;

    c_out.send(a_in+b_in) ;
    d_out.send(a_in-b_in) ;
};



array ... アレーの構造の記述
```

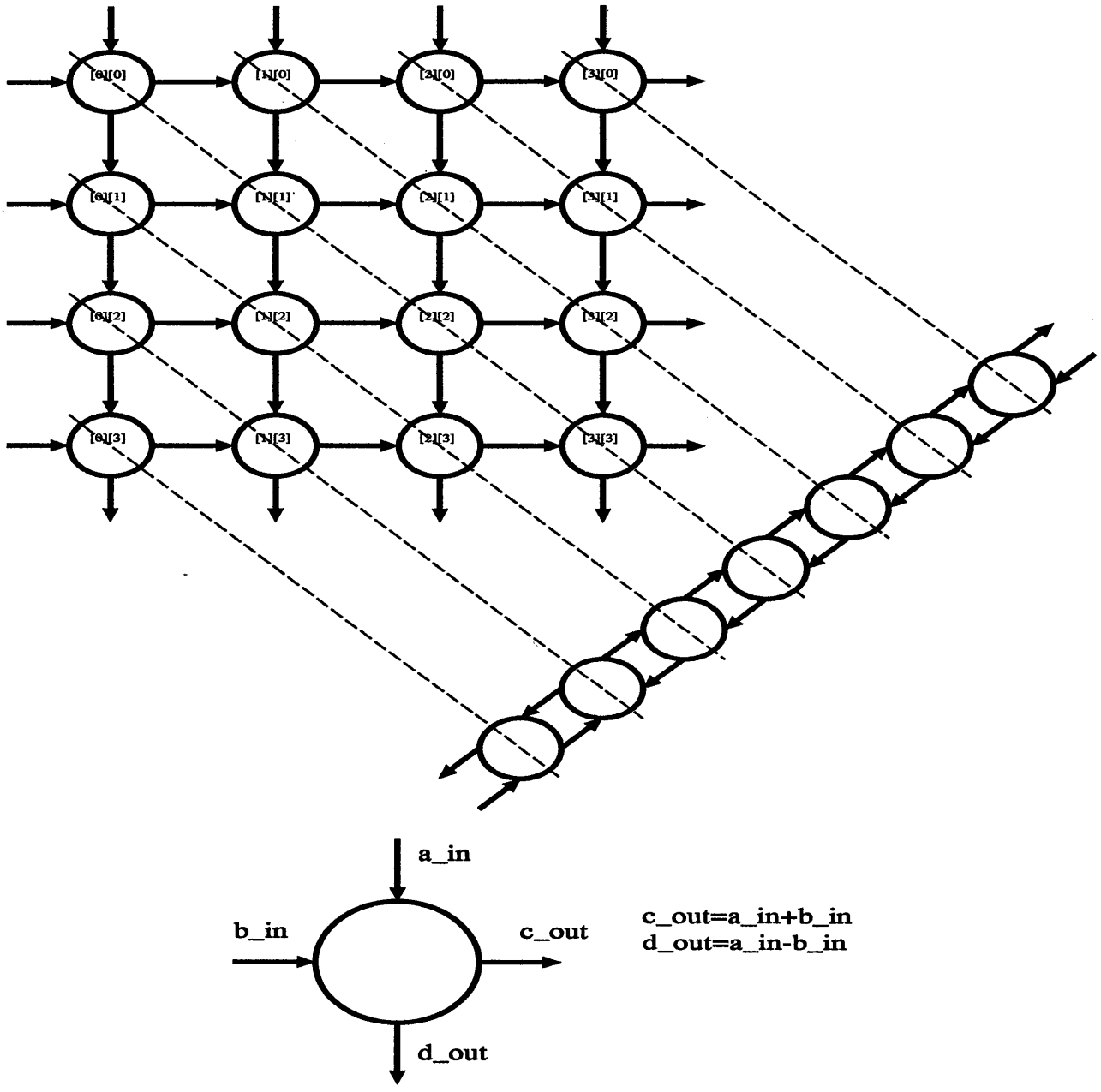

Figure 5.1: 動作確認のために入力する仕様ファイルの図的表現

```
{
    cell[4][4] ;

    outport out1 ;  ... アウトポートの宣言
    outport out2 ;
    outport out3 ;
    outport out4 ;
    outport out5 ;
    outport out6 ;
    outport out7 ;
    outport out8 ;

    inport cell[0][0].a_in ;  ... インポートの宣言
    inport cell[0][0].b_in ;
    inport cell[0][1].b_in ;
    inport cell[0][2].b_in ;
    inport cell[0][3].b_in ;
    inport cell[1][0].a_in ;
    inport cell[2][0].a_in ;
    inport cell[3][0].a_in ;



    for(i=0;i<3;i=i+1)        ... 結線の定義
        for(j=0;j<3;j=j+1)
        {
            cell[i][j].d_out-cell[i][j+1].a_in ;
            cell[i][j].c_out-cell[i+1][j].b_in ;
            cell[i+1][j].d_out-cell[i+1][j+1].a_in ;
            cell[i][j+1].c_out-cell[i+1][j+1].b_in ;
        }


    cell[0][3].d_out-out1 ;  ... アウトポートの結線の定義
    cell[1][3].d_out-out2 ;
    cell[2][3].d_out-out3 ;
    cell[3][3].d_out-out4 ;
    cell[3][0].c_out-out5 ;
    cell[3][1].c_out-out6 ;
    cell[3][2].c_out-out7 ;
```

```
    cell[3][3].c_out-out8 ;

};
```

5.2 シミュレートの結果

前節の仕様ファイルをSpasaに入力し、その出力をコンパイルして得られたシミュレートを行う実行ファイルに以下の入力データを入力してシミュレートした。

```
cell[0][0].a_in ... 1 1
cell[0][0].b_in ... 2 2
cell[0][1].b_in ... 3 3
cell[0][2].b_in ... 4 4
cell[0][3].b_in ... 5 5
cell[1][0].a_in ... 6 6
cell[2][0].a_in ... 7 7
cell[3][0].a_in ... 8 8
```

これは、図で描くとFigure 5.2のような状態になる。

5.2.1 射影なしの場合

射影しない場合は、以下のような結果を出す。

```
out1 :
  data: -13 time: 8
  data: -13 time: 10
 out2 :
  data: 4 time: 9
  data: 4 time: 11
 out3 :
  data: -6 time: 10
  data: -6 time: 12
 out4 :
  data: 5 time: 11
  data: 5 time: 13
 out5 :
  data: 24 time: 4
```

```
  data: 24 time: 6
 out6 :
  data: -5 time: 6
  data: -5 time: 8
 out7 :
  data: -17 time: 8
  data: -17 time: 10
 out8 :
  data: -15 time: 10
  data: -15 time: 12
```

よって、最後に出力されるデータは、時刻 13 に out4 から出力されるデータなので、この計算時間は 13 ということがわかる。

5.2.2 (1,-1) 方向に射影した場合

結果は同じで、計算時間は 15 となった。この方向には射影可能ということになる。

5.2.3 (1,1) 方向に射影した場合

結果は以下のようになった。

```
out1 :
 empty
out2 :
 empty
out3 :
 empty
out4 :
  data: -32 time: 8
  data: -32 time: 10
  data: -12 time: 12
  data: -12 time: 14
  data: 24 time: 16
  data: 24 time: 18
  data: 24 time: 20
  data: 24 time: 22
 out5 :
 empty
out6 :
```

```
 empty
out7 :
 empty
out8 :
  data: -20 time: 7
  data: -20 time: 9
  data: -44 time: 11
  data: -44 time: 13
  data: -48 time: 15
  data: -48 time: 17
  data: 8 time: 19
  data: 8 time: 21
```

これは明らかに射影しない場合と結果が異なる。よって、この方向の射影はできないことがわかる。

5.2.4 (2,-1) 方向に射影した場合

これは配線が局所的でなくなってしまう。結果は前のまま (各入力ポート 2 つ) の入力データでは射影なしの場合と結果も計算時間も変わらなかった。これはこの入力データでは、射影によって 1 つになるセルが同時には計算が行われていないことを示している。
入力データを各入力ポート 3 つにしたら、計算時間は、射影なしの場合は１７、(2,-1) 方向に射影した場合は２０となった。(計算結果は同じ。)

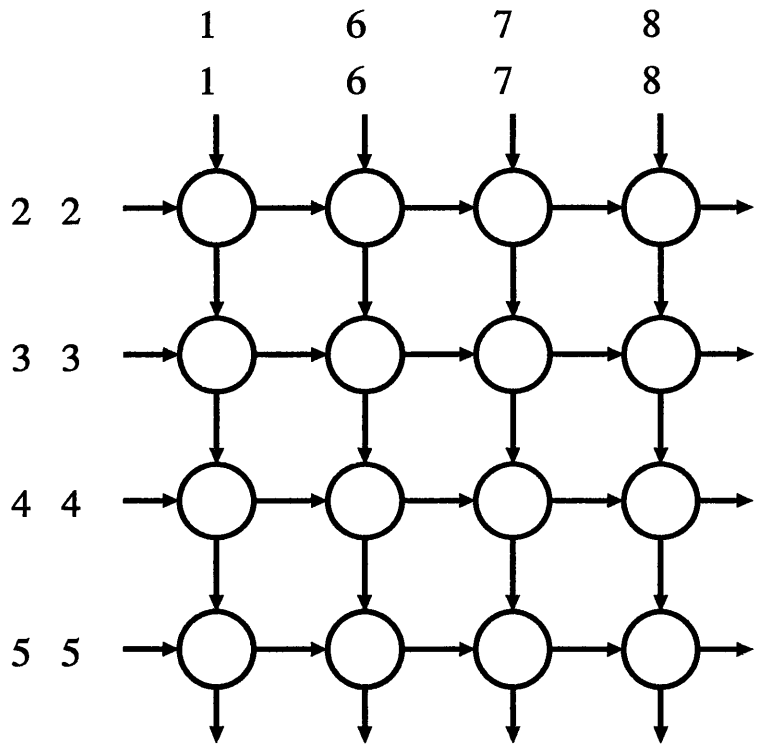

Figure 5.2: シミュレートの初期状態

第 6 章

まとめと今後の課題

本研究では動作確認が難しい非同期式シストリックアレーのシミュレートと設計の際に重要となる射影を行う Spasa を作成した。
今後の課題としては、現在の射影ベクトルの制限を外すことがあげられる。
また、シストリックアレーではなくなるが、セルの構造を、一様ではなくしてしまい複雑にすることにより、複数のコンピュータがつながっている (コンピュータネットワーク) ような状態のシミュレーションなどにもつながると思われる。

謝辞

本研究を行うにあたり、全般的な御指導、御助言を戴いた東北大学工学部 阿曽弘具教授、成富敬助手に心から感謝致します。
また、日頃から本研究の御指導、御意見を賜わった並列グループの藤岡豊太氏、佐藤英氏、菅谷至寛氏に深く感謝致します。
また、プログラミングについて御享受を戴いた後藤英昭氏、鈴木基之氏に感謝の意を表します。
最後に、日頃よりお世話になった阿曽研究室の皆様に心より感謝致します。

参考文献

[1] S.Y.Kung *et.al.:* "Wavefont Array Processor : Language,Architecture and Application", *IEEE Trans.Computers,* Vol.C-31,No.11.pp.1054-1065 (Nov.1982).

[2] S.Y.Kung :"VLSI ARRAY PROCESSORS", SYSTOLIC ARRAYS Edited by Will Moore *et.al.* pp.7-24,1987.

[3] 阿曽弘具: "シストリックアレーの自動設計法" , 電気情報通信学会論文誌 (D)、Vol.J71-D,No.8, pp.1487-1495.

[4] 佐藤健一: "シストリックアルゴリズム実現と検証に関する研究", 東北大学大学院情報工学修士論文.