非同期並列処理系の設計開発支援システム

佐藤健一[†,☆] 阿曽弘具[†]

情報処理の高速化を実現するために種々の並列処理アーキテクチャが考えられている．本論文では，並列処理アーキテクチャとして非同期動作するものを対象に，シストリックアレーと同様の一様な構造をもつ非同期処理アレーを定式化し，その仕様記述言語 a-MSDL を与える．さらに，非同期処理アレーを設計する組織的方法を与え，その設計作業を支援するために構築した設計開発支援システムについて述べる．非同期処理アレーはデータ駆動で動作するため大域的なクロックを必要としないという特徴をもつ．仕様記述言語はモジュール単位の記述で要素プロセッサの記述と結線構造の記述を分離しており，ハードウェアによる実現のための厳密な記述を可能にしている．また，シミュレータを自動的に構成する方法が得られる．組織的設計法は，問題の解法を多重ループプログラムで与えることにより非同期処理アレーを構成するもので，プログラムの等価変換に基礎をおいている．設計はまずプリミティブアレーと呼ぶ非同期処理アレーに変換し，そのプロセッサ空間を射影法に基づいて無駄のないプロセッサ割当を行うものである．構築した支援システムは，プログラム変換系 PriTran，シミュレータ生成系 SimGen，対話的ジェネリックシミュレータ実行系 SimRun からなる．これらは，組織的設計法を半自動的に実行するもので，より高性能な非同期処理アレーの設計を容易にしている．

A Design Support System for Asynchronous Parallel Processor Arrays

KEN'ICHI SATO [†,☆] and HIROTOMO ASO [†]

Parallel processing architectures are known to realize speedup for many complicated or massive information processings. In this paper, an architecture realizing asynchronous behavior of parallel processing is proposed as one with uniform structure as systolic arrays, which is called asynchronous processing array. For the asynchronous processing array architecture, a specification language a-MSDL is proposed. A systematic design method for asynchronous processing arrays is developed and it is realized in a design support system. Those are described in this paper. An asynchronous processing array is of a data-driven architecture and global clocks are not necessary. The specification language provides a separated description of element processors and their interconnections. This language gives a strict description so that it becomes easy to implement a hardware and its simulators can be produced automatically. The systematic design method is based on equivalent transformations between programs. It works for nested loop program specifications of problems to be solved and produces a primitive array (a kind of asynchronous processing array), and by projection method, possible arrays are produced. The proposed design support system consists of a program transformer PriTran, a simulator generator SimGen, and an interactive generic-simulator executer SimRun. These makes us easy to design effective asynchronous processing arrays.

1. はじめに

大量の情報の処理や複雑な情報処理の要求に応えるために処理の高速化が必要不可欠であり，アルゴリズムの工夫や計算素子の高速化がなされると共に，処理の並列化が考えられてきた．並列処理アーキテクチャの1つにシストリックアレーがある．これは演算の並列性・規則性に優れ，通信の局所性，構造の一様性・拡張性をもち，VLSI として実現することに適している．実験的には既に VLSI 素子が実現されており，より広範な実用的な利用のために，その設計技術，利用技術の開発が望まれている．

本論文は，シストリックアレーに非同期動作を導入した非同期処理アレーの組織的設計法を与えることを目的とする．その設計法に基づいて設計開発支援システムを構築し，その概要を述べる．設計支援システム

† 東北大学工学部通信工学科
Department of Communication Engineering, Faculty of Engineering, Tohoku University
* 現在，ソニー株式会社
Presently with Sony Co. Ltd.

は設計法の正しさ，対象とする問題の一般性（特定の問題ではないこと）で評価される．このため，非同期処理アレーを一般的な形で定式化し，その仕様記述言語 a-MSDL を提案する．また，一般的形で与えられた問題からそれを設計する方法を与え，その手順の正しさを示す．この設計法は，従来の同期処理アレーの設計法を拡張したものであり，アレー空間への射影変換を陽に与える点で新しいものである．この考え方はシストリックアレーの設計法にも適用でき有用である．

シストリックアレーは H. T. Kung により行列積の並列計算機構として提案された[3]（同様の計算機構はそれまでにも考えられていた[1),2)]が，魅力的な名前の提案によりよく知られるようになった）．シストリックアレーは同期的に動作するものとして考えられたが，そのために，データ流のすれ違いが起こり，ある時点でみると待機状態にあるプロセッサが多数存在することがある．非同期動作をさせることによってその無駄をなくし動作時間の短縮が可能となり，プロセッサの有効活用が可能となる．また，大規模なシストリックアレーでは同期のためのクロックの配布に工夫が必要となり，それを必要としない非同期並列処理系は製造上の利点をもつ．本論文では，シストリックアレーと同様に同一種類のプロセッサからなり一様な結線構造をもつ非同期並列処理系を定義している．そのような処理系として既にウェーブフロントアレー[4]が知られており，実際のハードウェアが作られ，それを利用するための設計法も議論されている．これは結線構造が2次元メッシュ構造に限定されたものであり，本論文ではこれを多次元一般格子構造に一般化している．

ウェーブフロントアレーを設計するための言語 MDFL[4] はアレー内の計算をデータ流の波頭の処理として記述するもので，プロセッサと結線構造を陽に記述するものとはなっていない．本論文で提案する仕様記述言語はアレー自体の仕様を記述するものである．仕様記述言語を与えることにより，図を用いるなどの発見的な思考を必要とするものに較べて，より組織的な設計を可能にし，ハードウェアの設計を容易にする．

同期動作をするシストリックアレーの設計法については各種のものが知られている[5)~11)]．本論文で提案するものは多重ループプログラムから a-MSDL による記述に変換する方法であり，文献[5),8),9),11)]などの方法を非同期処理アレーの設計に拡張したものである．アレーの配置空間を求める方法に射影ベクトルを基づく方法[9),11)]があり，本論文でも採用している．文献 11）の射影方法では射影先が整数ベクトルになるようにするためには発見的な思考を必要とするが，本論文では自動的に整数ベクトルになるように改良している．

以下では，2章で非同期処理アレーを定義し，仕様記述言語を与え，それをもとに開発したシミュレータについて説明する．3章では，問題の解法の記述として与える多重ループプログラムについて述べ，それをもとに非同期処理アレーを設計する方法を与える．4章では，その設計法に基づいた設計開発支援システムの概要とその実行例を示す．

2. 非同期並列処理系

2.1 非同期処理アレー

非同期並列処理系として，同一の要素プロセッサを格子空間の各点に配置し，方向を持つ局所的な通信線だけによって結合した非同期処理アレーを考える．要素プロセッサをセル，格子空間を配置空間といい，各点の座標をアドレスと呼ぶ．各セルはアドレスによって区別され，アドレス α のセルをセル α とも呼ぶ．

各セルは入力ポートと出力ポートをもち，それらによって外部ないし他のセルと接続される．入力ポートは通信用バッファ（FIFO キュー）から構成されている．出力ポートはレジスタであるとする．セルの機能は，入力ポートからの受信操作（通信用バッファからのデータの取り出し），出力ポートの値を入力ポートの値から定める計算処理，出力ポートからの送信操作（接続先の入力ポートの通信用バッファへの格納）からなる．計算処理はセル自身がもつ固有のクロックに従って，通信用バッファが空でないときはいつも実行されるものとし，バッファが空になると一時停止する（待機状態）．すなわち，データ駆動型の機構であり，外部からのグローバルなクロックは存在しないものとする．通信用バッファは任意の個数のデータを格納できるものとし，送信操作によってあふれは生じないものと仮定する（実際的には通信用バッファは有限であり，それが満杯のときは送信操作はそれが空くまで待つ機能を必要とするが，簡単のため上の仮定をおいた）．

セル間の結合は，入出力ポート間の結合として定める．入力ポートの接続先の出力ポートは唯一である（出力ポートは複数の入力ポートに接続されうる）．アレーとしての外部入力のために，いくつかのセルのある入力ポートは外部端子と同一視される．そのような入力ポートが出力ポートからの接続をもつとき，外部入力は通信用バッファへの初期設定としてのみ与えるものとする．

2.2 仕様記述言語

上に述べた非同期処理アレーを記述するものとして，仕様記述言語：a-MSDL（asynchronous Module

Specification Description Language) を与える．この言語の文法はコンパイラコンパイラ yacc の入力仕様として定められているが，ここでは，その詳細説明の煩雑さを避けるため，文法に従って記述した例をもって文法の説明にかえる．

a-MSDL は，セルをモジュールとして記述し，セル間結線も一つのモジュールの内部構造として記述する．セルの記述例を**図1**に示す．//以下は注釈である．セル自体は次の形の module 宣言文で定義される．

module セル名 {セル機能記述}

セル機能記述は入出力ポートの宣言と処理操作の記述からなる．入出力ポート名は，inport，outport 宣言文で宣言する．処理操作は，for () を前置した複合文で記述し，複合文の中には次のものを記す．

入力ポート名．recv ()

出力ポート名．send (計算式)

前者が受信操作，後者が計算処理と送信操作を表す．

```
module rc {
  inport win,xin,yin;          //入力ポートの宣言
  outport wout,xout,yout;      //出力ポートの宣言
    for(){
      win.recv();              // win から入力
      wout.send(win);          // wout へ win を出力
      xin.recv();              // xin から入力
      xout.send(xin);          // xout へ xin を出力
      yin.recv();              // yin から入力
      yout.send(yin+win*xin);
                               // yout へ計算結果を出力
    }
  }  // Positions of ports are not specified.
```

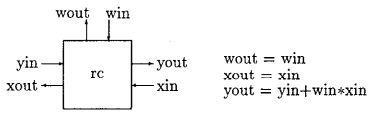


図1 セルの記述
Fig. 1 Description of cell.

```
const N=3;
module cor {
  module rc[N];                        //使用するセルの宣言
  external inport Win[N],Xin[N],Yin;   //外部入力ポートの宣言
  external outport Yout;               //外部出力ポートの宣言
    for(j=0<N){
      Win[j]==rc[j].win;       // j 番目の Win は j 番目の rc の win と等価
      rc[j].win <- rc[j].wout;  // j 番目の rc の win は同じ rc の wout に接続
      Xin[j]==rc[j].xin;
      if( j!=N-1 ) rc[j].xin <- rc[j+1].xout;
      if( j==0 ) Yin==rc[j].yin;
      else    rc[j].yin <- rc[j-1].yout;
      if( j==N-1 ) Yout==rc[j].yout;
    }
  }
```

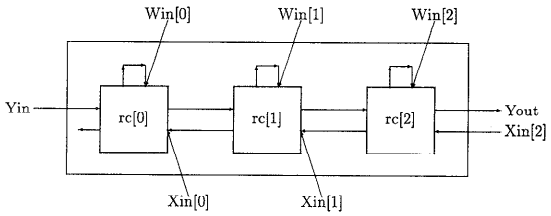


図2 アレー構造の記述
Fig. 2 Description of array structure.

このセルを用いたアレー構造は，**図2**の下部に示す結線に対して上部に示すように記述される．以下ではアレーモジュールとも呼ぶ．const は記述の中で用いる定数に名前を与える．アレーで用いるセルは通常の言語の配列宣言と同じ形で次のように宣言する．

module セル名[定数][定数]；

(これは配置空間が2次元の場合である．) このセルを子モジュールと呼ぶ．入出力ポート宣言の external の指定は，その名のポートが外部端子であることを指定し，それは子モジュールの入出力ポートのどれかの別名として外部に見えることを指示する．その対応は

外部ポート名＝＝子モジュール名．ポート名

で記述する．これらはアレー自体の入出力ポートの記述を与えている．子モジュールの入出力ポート間の結線は，

モジュール名[アドレス]．入力ポート名

＜－モジュール名[接続先アドレス]．出力ポート名

として定められる．ただし，アドレスは単純変数または整数を四則演算で結合した式である．単純変数自体は，for 文の中で宣言する．また，アドレスの違いにより接続先が異なる場合，if 文を用いる．これは，境界をもつアレーと外部との接続を記述するために必要な機能である．

子モジュールとしてアレーモジュールをとることも可能で，階層的な結線構造も表現できるが，以下では，単純な子モジュールを結合したアレー構造のみ考える．また，アレー内に2種類以上のセルの記述も可能で，数種のセルを結合して得られる実際的なマルチプロセッサシステムの記述法にもなっている（理論的には，複数のセルをもつアレーは，単一のセルが複数の機能を持ち内部でそれらを切り替えるということによって，単一のセルからなるアレーとみなせる)．

a-MSDL は，セル間結線とセルの記述を分離し，アレー全体の構造をわかりやすくしているところに特徴がある．従来のシストリックアレーなどの一様構造の記述にみられたセルの記述の中に入出力ポートの通信の方向を示して，セル間結線の記述とするものはアレーの境界における記述を面倒にし，理解しにくくしていた．

2.3 シミュレータ

非同期処理アレーは，その動作が非同期であるためその処理時間などの性能を仕様から解析的に評価することが難しい．そこで，シミュレータを用いて評価することを考えた．しかし，非同期動作であるために単純な繰返し処理では実現できない．

シミュレータの動作原理は次のようにした．

（1） アレーの構成要素であるセルを実行待ち行列に並べる．外部入力データを入力する．

（2） 待ち行列の先頭から一つのセルを取り出す．それが入力待ち状態（どこかの入力ポートの通信用バッファが空である状態）であれば，それを計数し，待ち行列に戻し，次のセルを取り出し同様のチェックをする．その計数値から，すべてが入力待ち状態であればシミュレーションを停止する．取り出したものが入力待ち状態でなければ計数を0にリセットして手順（3）へ．

（3） そのセルを入力待ち状態になるまで動作させる（この動作中，他のセルの入力ポートへデータが送られるので，いくつかのセルの入力待ち状態を解消する）．入力待ち状態になったら手順（4）へ，入力待ち状態になる前に処理が終了した場合は手順（5）へ．

（4） 入力待ち状態になったセルを実行待ち行列の最後に並べる．

（5） 実行待ち行列にセルがあれば，手順（2）に戻る．なければ，終了する．

この手順（3）を実行中，演算回数と送受信操作の回数を計数し，処理の終了時に出力させる．

上の手順ではセルの動作に注目しているので，アレー全体での処理時間はこのままではわからない．そこで，タイムスタンプ付きデータ型を導入し，各送受信データに，セルの演算時間を単位時間とするタイムスタンプをつけるようにした．このためのセルの動作は次の通りである．

各セルは固有の時計をもち，その時刻を T_c とする．

（1） 受信操作では，時刻 T_c とデータのタイムスタンプ T_d を比較し，$T_c < T_d$ であれば $T_c = T_d$ と更新し，$T_c \geqq T_d$ であれば T_c は変更しない．（前者の場合，セルがデータがくるのを待っていたことを意味し，後者の場合データが処理を待っていたことを意味する．）

（2） 計算処理では，T_c を単位時間進める．

（3） 送信処理では，T_c の値をタイムスタンプとして出力データにつける．

これにより最終的に出力されたデータのタイムスタンプを見ることによってアレー全体での処理時間を評価できる．

仕様記述言語 a-MSDL で記述された非同期処理アレーに対して上に述べた動作をするシミュレータを実現するシミュレータ生成系 SimGen (Simulator Generator) を開発した．これは，与えられた非同期処理アレーのシミュレータを言語C++で記述されたプログラムとして生成し，自動的にコンパイルして，実行形式のシミュレータを生成する．これにより，設計した非同期処理アレーの動作を確認し，処理時間などの性能を評価することを可能にした．

3. 組織的設計法

3.1 標準形プログラム

処理したい問題に対して，非同期処理アレーをいかに設計するかについて考える．

まず，処理したい問題の解法が，n 個の（ループ）制御変数をもつ多重ループプログラムで，その本体は割当文だけからなり，内部ループの外には実行文がないという形で与えられているとする．この形のプログラムを標準形プログラムと呼ぶ．この仮定は，本体に if 文があったり，内側のループの直前，直後にも初期設定や後処理の文があるようなより一般的なループプログラムから，この形への変換が可能であり[10]，一般性を失わない．また，要素プロセッサであるセルは他のセルの出力ポートの値を入力ポートで受け取り，それをもとに出力ポートの値を計算している．そこで出力ポートを変数として記述すると転送・計算のセルの動作はその変数を用いた割当文の列として記述できる．従って，アレーの動作はその割当文の列を時空間にわたって繰り返すことであるといえる．このことはその動作の記述が標準形プログラムによる記述に対応すると考えられ，標準形プログラムは設計仕様として自然なものと考えられる．

標準形プログラムの一般形は次のように表せる．

```
for (j1=b1 to e1)
  .....
  for (jn=bn to en)
   {a1[k1(J)]
      =f1(J, <a1^q[k1^q(J)]|q∈Occur>);
    ...
    ap[kp(J)]
      =fp(J, <ap^q[kp^q(J)]|q∈Occur>);
    ...
   }
```

ここで，$J=(j_1, \cdots, j_n)$ は制御変数の組であり，その他の記号の意味は次のとおりである．

b_1, e_1 等：各制御変数の上下限の値，

a_p：p 番目の文で値が更新される配列名，

a_p^q：p 番目の文で値が参照される q 番目の配列名，

$k_p(J)$, $k_p^q(J)$：それぞれの配列要素を指示する添え字式，

f_p：p 番目の文の値の計算式，

Occur：参照変数の出現順 q の集合．

配列変数 $a_p[k_p(J)]$ は右辺の計算式 $f_p(\cdots)$ により値が更新され，更新変数と呼ぶ．計算式の中の＜…＞はその式の中で使われている配列変数 $a_p^q[k_p^q(J)]$ のリストであり，その変数を参照変数と呼ぶ．配列でない通常の変数は添え字式が定数である配列変数として扱う．添え字式は制御変数と定数からなる算術式，計算式は if-then-else 関数と四則演算からなる式とする．制御変数の値の更新は 1 ずつ増えるものとする．

制御変数のとる値の集合を Ind と記し，インデックス集合という．

$$Ind=\{(j_1, \cdots, j_n) | b_1 \leqq j_1 \leqq e_1, \cdots, b_n \leqq j_n \leqq e_n\}$$

この元を制御変数と区別するため，$\mathbf{j}=(j_1, \cdots, j_n)$ と記す．$\mathbf{j}$ は列ベクトルとみなす．

この標準形プログラムは逐次実行されるものであり，そのことから，本体の各文の計算には依存関係が生じる．すなわち，制御変数 J がとる具体的な値 $\mathbf{j}$ における計算で使われる参照変数 $a_p^q[k_p^q(J)]$ は実際の添え字 $k_p^q(\mathbf{j})$ における値 $a_p^q[k_p^q(\mathbf{j})]$ をとっているが，その配列名 a_p^q が更新変数の配列名 $a_{p'}$ と同じ場合，その値は，$k_{p'}(\mathbf{j}')=k_p^q(\mathbf{j})$ となる最近の過去の $J=\mathbf{j}'$ で計算されて更新されているはずである．従って，$\mathbf{j}$ における計算は $\mathbf{j}'$ における計算の後でなければならないという依存関係がある．これを Ind 上の計算依存関係と呼ぶ．p' と $\mathbf{j}'$ は p, q, $\mathbf{j}$ から一意に定まる．これを $st(\mathbf{j}, <p, q>)=p'$, $\mathbf{j} \lhd <p, q>=\mathbf{j}'$ と表し，それぞれ，文指定関数，逆行関数と呼ぶ（詳しくは文献 8) 参照）．実際の求め方については後の例を参照）．$\mathbf{j}-\mathbf{j} \lhd <p, q>$ が $\mathbf{j}$ に依存しない定数 d_{pq} になるとき，d_{pq} をデータ依存ベクトルと呼ぶ（計算依存ベクトルと呼ぶべきであるが歴史的にこのように呼ばれている）．

以下では，すべての参照変数についてその逆行関数がデータ依存ベクトルにより定まるプログラムを対象とする．零ベクトルでない数値ベクトルとして異なるすべてのデータ依存ベクトルが m 個あるとして，その組を $n \times m$ 整数行列 D で表し，データ依存行列と呼ぶ．

参照変数 A［k(J)］の配列名 A が更新変数の配列名として現れていないとき，逆行関数は定義されない．これはこのプログラムから非同期処理アレーを構成する際に不都合となる．そこで，そのような場合，

A[k(J)]＝A[k(J)]

という割当文をループ本体の最初につけ加える．このようにしても，逐次処理プログラムとしての計算に影響を与えないことは明らかである．このプログラムの等価変換を流れ化変換（pipelining transformation）という．

D の求め方について例を示す．

【例 1】次の相関係数を計算するプログラムを考える．

```
for (i=0 to M-1)
  for (j=0 to N-1)
  {Y[i]=Y[i]+W[j]* X[i+j];}
```

参照変数 W[j]，X[i+j] に関して流れ化変換をする．本体は次のようになる（以下での説明のため，文番号を付け，各添え字式を一般形で用いた記号を使って表す）．

```
1: W[k1(i, j)]=W[k1^w(i, j)];
2: X[k2(i, j)]=X[k2^x(i, j)];
3: Y[k3(i, j)]=Y[k3^y(i, j)]+W[k3^w(i, j)]*
                X[k3^x(i, j)];
```

ここで，参照変数の出現順 q は配列名の小文字で示した．各添え字式は次のように定義される．

$k_1(i, j)=k_1^w(i, j)=k_3^w(i, j)=j$

$k_2(i, j)=k_2^x(i, j)=k_3^x(i, j)=i+j$

$k_3(i, j)=k_3^y(i, j)=i$

これから逆行関数を求める．$\mathbf{j}=(i, j)$ である．文 1 について，$(i, j) \lhd <1, w>=(i', j')$ となる i', j' を求める．

$k_1(i', j')=j'=k_1^w(i, j)=j$ かつ

(i', j') は (i, j) より過去

から，$(i', j')=(i-1, j)$ を得る．同様に文 2 について，

$k_2(i', j')=i'+j'=k_2^x(i, j)=i+j$

から，$(i, j) \lhd <2, x>=(i', j')=(i-1, j+1)$ を得る．文 3 については，$(i, j) \lhd <3, y>=(i, j-1)$, $(i, j) \lhd <3, w>=(i, j) \lhd <3, x>=(i, j)$ を得る．従って，

$$D=\begin{array}{c} \\ i \\ j \end{array}\begin{array}{c} \begin{array}{ccc} w & x & y \end{array} \\ \begin{bmatrix} 1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \end{array}$$ となる．

3.2 プリミティブアレー

標準形プログラムに対して，インデックス集合 Ind の元をアドレスとするセルを用意し，その計算機能をループ本体と等しく定める．参照変数で，その配列名，添え字式が同じものに一つの入力ポートを割り当て，更新変数にはそれぞれに対して出力ポートを割り当てる．入出力ポート間の結線は計算依存関係に基づいて行う．すなわち，参照変数 $a_p^q[k_p^q(J)]$ に対して，st

(**j**, <p, q>)=p', **j** ◁<p, q>=**j**' のとき，セル**j**の入力ポート<p, q>（の同値類）をセル**j**'の出力ポートp'に接続する．このようにして得られる非同期処理アレーをプリミティブアレーと呼ぶ．

プリミティブアレーにおいては，すべての入力ポートにデータがそろっているセルから動作を開始し，その出力データはデータ依存ベクトルの方向に流れる．これはセルの動作順序がデータ依存ベクトルから自然に定まることを意味する．また，各セルは1度しか動作しないので，このアレーがプログラムが意味する機能を実現していることは明らかである．複数のセルが同時に動作することがあるので処理時間は短縮される．

プリミティブアレーは従来，データ依存グラフ[1),2),6)]と呼ばれていたものを非同期並列処理系として実現したものに相当する．また，RIA (Regular Iterative Array)[7)]の非同期実現であるともいえる．

与えられた標準形プログラムから対応するプリミティブアレーの仕様記述を求める方法を与える．まず，その方法を例1のプログラムに対して適用して説明する．構成するアレーをcor，セルをrcと名付ける．各配列名に対応して，新しい配列名win，wout，xin，xout，yin，youtを導入し，もとのプログラムと等価な次の本体をもつプログラムを作る．

```
if  i==0  then win[i, j]=W[j];
          else win[i, j]=wout[(i, j)-(1, 0)];
if  i==0‖j==N-1
      then xin[i, j]=X[i+j];
      else xin[i, j]=xout[(i, j)-(1,-1)];
if  j==0  then yin[i, j]=Y[i];
          else yin[i, j]
          =yout[(i, j)-(0, 1)];
  wout[i, j]=win[i, j];
  xout[i, j]=xin[i, j];
  yout[i, j]=yin[i, j]+win[i, j] * xin[i, j];
if  j==N-1  then Y[i]=yout[i, j];
```

等価性は容易に確かめられる．このプログラムの中ほどのif文を含まない割当文からモジュールrcを作る．この結果が図1に示すものである．また，その前後のif文からアレーモジュールcorを図3のように定める．いずれもその定め方が文の変換により実現されていることが理解されよう．

この変換アルゴリズムを一般的な形で示す．まず，いくつかの記法を導入する．参照変数 $a_p^q[k_p^q(J)]$ に対して，$a_p^q=A$，$k_p^q=k$，**j** ◁<p, q>=**j**−d として，集合：Init(A, k)={**j**|**j**−d∉Ind} ⊆ Ind

```
module cor {
  module rc[M][N];
  external inport Win[N],Xin[M+N],Yin[M];
  external outport Yout[M];
    for(i=0<M)
      for(j=0<N){
        if( i==0 ) Win[j]==rc[i][j].win;
        else rc[i][j].win <- rc[i-1][j].wout;
        if( i==0 || j==N-1 ) Xin[i+j]==rc[i][j].xin;
        else rc[i][j].xin <- rc[i-1][j+1].xout;
        if( j==0 ) Yin[i]==rc[i][j].yin;
        else rc[i][j].yin <- rc[i][j-1].yout;
        if( j==N-1 ) Yout[i]==rc[i][j].yout;
    }
  }
```

図3 プリミティブアレー
Fig. 3 Primitive array.

を定める．これは，配列Aのデータが初期値として与えられるプリミティブアレーのセルアドレスを示す．

例1では次のようになる．

$\mathrm{Init}(W, k_1^w)=\{(0, j) \mid 0\leqq j<N\}$

$\mathrm{Init}(X, k_2^x)=\{(0, j), (i, N-1) \mid 0\leqq j<N, 0\leqq i<M\}$

$\mathrm{Init}(Y, k_3^y)=\{(i, 0) \mid 0\leqq i<M\}$

更新変数 $a_p[k_p(J)]$ に対して，$a_p=B$，$k_p=k'$ として，次の集合を定める．

Last(k')= {**j**|各$\mu\in$k'(Ind) について **j**=max{**j**' | k'(**j**')=μ}}⊆Ind

ここで，maxは辞書式順序での最大であり，k'(Ind)は配列Bの添え字の集合である．この集合は，配列Bの最終的結果を出力するセルアドレスを示す．例1では，$k_3(i', j')=\mu$ ならば $i'=\mu$ より次のようになる．

$\mathrm{Last}(k_3)=\{(\mu, N-1) \mid 0\leqq\mu<M\}$

変数Jに対して，J∈Init(A, k)，J∈Last(k') なる論理述語を境界条件という．この条件は，実際に変換を適用するプログラムにおいて，より簡単な条件として記述できる（最悪の場合，各元$\mathbf{j}_t$をもとにしたj=$\mathbf{j}_t$なる述語を||(or演算)でつなげばよい）．例1については，次のようになる．

$(i, j)\in\mathrm{Init}(W, k_1^w) \Leftrightarrow i==0$

$(i, j)\in\mathrm{Init}(X, k_2^x) \Leftrightarrow i==0 \,||\, j==N-1$

$(i, j)\in\mathrm{Init}(Y, k_3^y) \Leftrightarrow j==0$

$(i, j)\in\mathrm{Last}(k_3) \Leftrightarrow j==N-1$

以上の記法を用いて一般的な方法を次に示す．

［プリミティブアレーへの変換］

（1） 与えられた標準形プログラムに対して，流れ化変換をして，データ依存ベクトルdをすべて求める．

（2） プログラム中の各配列名Aに対して，新しい2つの配列名ain，aoutを導入し，次の変換をする．

(2-1) 各参照変数A[k(J)]に対して，Init(A, k)を

求め，次の文を本体の最初におき，すべての文における A[k(j)]の出現を ain[J]で置き換える．

if J∈Init(A, k)
then ain[J]−A[k(J)]；
else ain[J]＝aout[J−d]；

(2-2) 各更新変数 B[k'(J)]に対して，それを bout[J]で置き換える．この結果，もとの割当文は次の形になる．

bout[J]＝f_p(…, ain[J], …)；

(2-3) 流れ化変換前の各更新変数 B[k'(J)]に対して，Last(k') を求め，次の文を本体の最後におく．

if J∈Last(k') then B[k'(J)]＝bout[J]；

(3) 手順 (2) で得られたプログラムを次のようにして a-MSDL による仕様記述に変換する．

(3-1) 子モジュールの名前を選び，子モジュールを記述する．その本体の記述は，次で得られる．

(a) すべての，ain について，次の文をおく．

ain. recv()；

(b) (2-2) に示す割当文を次の文に変換する．

bout. send(f_p(…, ain, …))；

(3-2) アレーモジュールを記述する．子モジュールの名前が cell のとき次のようにする．(2-1)，(2-3) の if 文に対して，次の文をおく．

if J∈Init(A, k)
then Ain[k(J)]＝＝cell[J]. ain；
else cell[J]. ain＜−cell[J−d]. aout；
if J∈Last(k')
then Bout[k'(J)]＝＝cell[J]. bout；

また，入力プログラムの for (…) は対応するアレーモジュールの for (…) に変換する．

各 **j**∈Ind に関する入力データ A[k(**j**)]はセル **j**∈Init(A, k) の入力ポートである外部ポート Ain[k(**j**)]に入力し，出力データ B[k'(**j**)]はセル **j**∈Last(k') の出力ポートである外部ポート Bout[k'(**j**)]から出力されることになる．

3.3 非同期処理アレーの設計法

プリミティブアレーをもとに実際の非同期処理アレーを構成する方法を与える．データ依存グラフからシストリックアレーを導く方法に射影による方法があり，その方法の原理を適用して，非同期処理アレーの構成法を導く．整数の集合を Z とすると，Ind⊆Z^nである．以下では，()をつけた数の並びは列ベクトルとする．

ある射影ベクトル $\pi\in Z^n$ ($\pi\neq 0$) を選び，Z^nをその方向に射影すると n−1 次元格子空間 Z^{n-1}が得られる．この π による射影をプリミティブアレーのある一つのセル $\mathbf{j}_0\in$Ind に $\mathbf{j}_0$から π の方向にある任意の格子点 $\mathbf{j}_0+s\pi$ ($s\in Z$) をアドレスとするセルにおける計算を割り当てることに対応させる．このとき，s の大小が計算の順序を反映するものとする．従って，1 未満の正の有理数 s' が存在して，$\mathbf{j}_0+s'\pi\in Z^n$となるような π は選ばないものとする．

さて，データ依存ベクトルを d とするとき，**j** での計算は **j**−d での計算の後でなければならない．d は **j** における参照変数，例えば A[k(**j**)]，について決まり，プリミティブアレーにおいてはその参照変数 A[k(**j**)]の値が更新されながら流れていく方向を示す．この値の流れはプリミティブアレーにおける時の流れを自然に定める．この時の流れと射影された空間における各セルの時の流れが一致しないとき，各セルの入力ポートに到着するデータの順序がもとのプリミティブアレーにおける順序と異なってしまう．従って，それぞれの流れの方向 d と π について，$\pi^T d\geqq 0$ が成立しなければならない (^{T}は転置．ベクトルの内積)．特に，セル **j** と **j**−d における計算が射影先の同じセルで行われる ($\pi=c\cdot d$) ならば，$\pi^T d>0$ (つまり，$c>0$) でなければならない．シストリックアレーの設計における射影ベクトルの選び方は，時間軸と直交しなければ任意であった[11]が，非同期処理アレーではこの制限がつくことが異なる点である．例 1 については，n−2 であり，$\pi^T=[u, v]$とおくと，$\pi^T D\geqq 0$ より，$u\geqq v\geqq 0$ が得られ，最も簡単な π は，$\pi^T=[1, 0]$である．

次に実際の計算を行うために選ばれたセルに新しいアドレスを与えることを考える．これを，Z^nから Z^{n-1}への線形写像 P として定める．P は(n−1)×n 整数行列として表現できる．すなわち，各 **j**∈Ind について，$\alpha-P\mathbf{j}\in Z^{n-1}$なるアドレスを対応させる．任意の s∈Z について，$\mathbf{j}_0+s\pi$ が $\mathbf{j}_0$に割り当てられることから，$P\pi=0$ が成立しなければならない．

この条件を満たす P を求める．まず，π の要素の非零要素を第 1 要素にする置換行列 R を選ぶ (R として π の要素をソートするものが実際的と思われる)．$R\pi=r=(r_0, r_1, r_2, \cdots, r_{n-1})$とする．定義から，$r_0\neq 0$ である．$S_j^2=\Sigma_{t=0}^{j} r_t^2$ とおき，次のように(n−1)×n 行列 Q を定める．

$$Q=\begin{bmatrix} -r_1 & r_0 & 0 & 0 & 0 & \cdots & \cdots & 0 \\ -r_2r_0 & -r_2r_1 & S_1^2 & 0 & 0 & \cdots & \cdots & 0 \\ \cdots & \cdots & & \cdots & & & & \\ -r_jr_0 & -r_jr_1 & \cdots & -r_jr_{j-1} & S_{j-1}^2 & 0 & \cdots & 0 \\ \cdots & \cdots & & & & \cdots & & \\ -r_{n-1}r_0 & -r_{n-1}r_1 & \cdots & \cdots & -r_{n-1}r_{n-2} & S_{n-2}^2 & & \end{bmatrix}$$

明らかに，Q は整数行列で，$Qr=0$ が成立する．そこで，

$P=QR$ と定める．$P\pi=QR\pi=0$ である．（この Q は，ベクトル r をベクトル $(S_{n-1}, 0, 0, \cdots, 0)$ に回転する変換行列 H を，2軸の回転を次々に求めてそれらをかけ合わせることによって求め，H の1行目を除いた部分行列に次の対角行列 K をかけることで得られる．$K=\mathrm{diag}(S_1, S_1S_2, \cdots, S_{j-1}S_j, \cdots, S_{n-2}S_{n-1})$．なお，H の第1行目は $(1/S_{n-1})\,r^T$，それを除く部分行列は $K^{-1}Q$ であり，$\det H=1$, $H^{-1}=H^T$ である．）例1では，R は単位行列でよく，Q は 1×2 行列で，

$$P=Q=[-v, u]=[0, 1]$$

となる．

上で求まった P を用いて，n−1 次元空間に射影された配置空間 V は次のようになる．

$$V=\{\alpha \mid \alpha=P\mathbf{j}, \mathbf{j}\in \mathrm{Ind}\}$$

この V を含む n−1 次元直方体のサイズは容易に定まる．プリミティブアレーにおける結線は，データ依存ベクトル d に関して，**j**−d から **j** へ接続する．従って，V においては，P**j**−Pd から P**j** へ接続しなければならない．すなわち，結線は，PD の各列ベクトルによって定まる．例1では，$PD=[0, -1, 1]$ となる．

境界条件を与えるための集合 Init(A, k) と Last(k') はそれぞれの元に P をかけたものとして得られる．これを P・Init (A, k)，P・Last (k') と記す．例1では次のようになる．

$$P\cdot \mathrm{Init}(W, k_1^w)=\{j\}$$
$$P\cdot \mathrm{Init}(X, k_2^x)=\{j\}$$
$$P\cdot \mathrm{Init}(Y, k_3^y)=\{0\}$$
$$P\cdot \mathrm{Last}(k_3)=\{N-1\}$$

この境界条件を記述している文の else 部の接続は，Init(A, k) の補集合の元に対する接続を意味しており，射影された場合もその補集合の元の射影先に接続する必要がある．この射影先の元の集合を $P\cdot \mathrm{Init}(A, k)^c$ と表す．例1では例えば次のようになる．

$$P\cdot \mathrm{Init}(W, k_1^w)^c=P\cdot\{(i, j) \mid i\neq 0\}=\{j\}$$

これは結果として then 部と同じものである．

V 上での各セルの動作は V 上の値をとる変数 I を用いて記述する必要がある．配列要素を指定する添え字式 k(J) に対応する I を用いた添え字式 $l(I)$ はデータ依存関係を保証する必要がある．これは与えられた $\alpha\in V$ に対して，$\mathbf{j}(\alpha)=\min\{\mathbf{j} \mid \alpha=P\mathbf{j}\}$ と定めたとき，$l(\alpha)=k(\mathbf{j}(\alpha))$ と定めることで達成される（これは依存ベクトルの定義から示されるが長くなるので割愛せざるを得ない）．k(J) が算術式で表されていれば，$l(I)$ も算術式で表すことができる．例1では，$\{\mathbf{j} \mid \alpha=[0, 1]\mathbf{j}\}=\{(i, \alpha)\}$ であるから，$\mathbf{j}(\alpha)=(0, \alpha)$ となり，$l_2^x(\alpha)=k_2^x(0, \alpha)=0+\alpha=\alpha$ である．同様に，$l_1^w(\alpha)=\alpha$，$l_3^y(\alpha)=0$.

以上をまとめて，次の設計法を得る．

［非同期処理アレーの設計法］

（0） 処理したい問題の解法を標準形プログラムとして記述する．

（1） 標準形プログラムからプリミティブアレーを求める．このとき，データ依存ベクトルの組 D が定まる．

（2） $\pi^TD\geqq 0$ となる整数ベクトル π を選ぶ．

（3） π から配置空間の変換行列 P を求める（$P\pi=0$）．

（4） プリミティブアレーの記述において，変数 $J=(j_1, j_2, \cdots, j_n)$ 変数 $I=(i_1, i_2, \cdots, i_{n-1})$ に，アレー内結線の記述に現れるデータ依存ベクトル d を Pd に，外部ポート Ain，Bout の添え字式 k(J) を $l(I)$ に置き換える．境界条件 J∈Init (A, k)，J∈Last (k') をもつ if 文については，I∈P・Init(A, k)，I∈P・Last(k') の条件に置き換え，else 部については，“if I∈P・Init $(A, k)^c$ then” という条件をもつ if 文に置き換える．それぞれの条件が常に真になる場合は条件部を削除する．また，子モジュールの個数を示す定数は，V を含む n−1 次元直方体のサイズで置き換える．

これにより，3重ループプログラムで表される解法は2次元の配置空間で，2重ループプログラムに対しては1次元の配置空間で，等価な動作をする非同期処理アレーを得ることができる．図3のプリミティブアレーに対して，例1に関して上に示してきたように，$\pi=(1,0)$ を選べば，$P=[01]$ となり，これにより上の方法で射影された配置空間をもつ非同期処理アレーが実は，図2に示すものであることがわかる．

ここで，注意しなければならないことは，一般的には V は Z^{n-1} 内ではとびとびの値をとっていることである．n=2 で π の要素の絶対値が1以下の場合は（整数として）連続した値をとることが保証でき，n=3 の場合もたいていの場合は連続にできる．しかし，一般の n の場合に連続に近くするにははさらに工夫が必要である．この詳細は稿を改めて議論する予定である．もう一つの問題点は，標準形プログラムでの割当文の計算式が制御変数 J に直接依存している場合，上に述べた設計法では非同期処理アレーを得ることができないことである（プリミティブアレーではアドレスとして **j** を参照できるので問題はない）．射影行列の導出において煩雑さをさけるためにこの点には言及しなかった．多くの紙幅を必要とするため詳細は割愛せざるを得ないが，そこで述べた回転行列 H に逆行列が存在することを利用してこの問題を解決できる．

4. 設計開発支援システム

4.1 システム構成

非同期処理アレーの性能（処理時間，必要なセル数など）は，上に述べた設計方法では，射影ベクトルの選び方に依存する．そこで，いろいろな射影ベクトルを試みることにより，設計を支援するシステムを構築した．設計開発支援システムは，**図4**に示すように，標準形プログラムからプリミティブアレーへの変換系PriTran，先に述べたシミュレータ生成系SimGen，次に述べるジェネリックシミュレータの実行系SimRunからなっている．

PriTranは，3.2節で述べた方法をコンパイラコンパイラyacc用いて実現している．ただし，アレーの結線構造の記述に必要な境界条件の構成は組み込んでいず，今のところ，プリミティブアレーの最終的記述を得るには人手が必要である．データ依存ベクトルは，標準形プログラムに添え字式の一致を判定させる機能を組み込んでそのプログラムを実際に実行させることにより求める方法をとっている．

SimGenは，プリミティブアレーの記述に対して，ジェネリックシミュレータを生成できる機能を付加してある．ジェネリックシミュレータとは，セルの配置空間に対してある射影ベクトルを与えるとその方向に射影された配置空間を求め，その配置空間をもつ非同期シストリックアレーのシミュレータとなるものである．これは3.3節に述べたことに基礎をおいている．

SimRunは，与えられた問題の入力データで典型的なものを用意しておき，可能な射影ベクトルを対話的に与え，それにより構成される非同期処理アレーのセル数を出力し，それを模擬実行し，処理時間などの性能情報を出力するものである．

4.2 実 行 例

例1について，プリミティブアレーのセル配置は**図5**（a）のようになる．格子点の数字はそこに置かれたセルの処理回数を示す．$\pi=(1,0)$とした場合のアレーについては既に示してきたが，SimRunによる性能評価は図5（b）のようになる．$\pi=(1,1)$とした場合を図5（c）に示す．処理時間はセル内処理が1単位時間であるとして示してある．

図6（a）の行列積プログラムについて，データ依存行列として図6(b)が得られ，$\pi \geqq 0$である任意のπが候補になる．πの要素を0,1に限った7通りの場合の結果を図6（c）に示す．どのπを選べばよいかが容易にわかり，設計を支援するシステムとなっていることがわかる．$\pi=(1,1,1)$の場合のセル配置を図6（d）

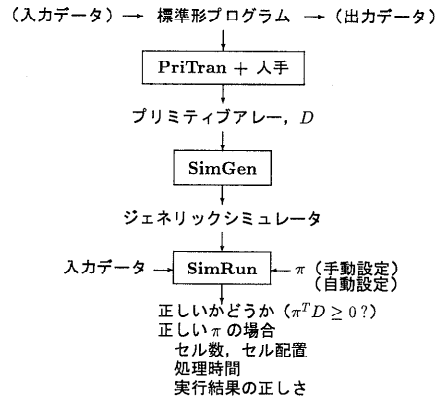


図4 設計開発支援システムの構成
Fig. 4 Design support system.

```
  j  0  1  2
 i+--+--+--+-
 0+  1  1  1
 1+  1  1  1
 2+  1  1  1
Num-of-cells: 9   Times: 7
(a) primitive array

      i j          w  x  y
P=i0| 0 1|, PD=i0| 0 -1  1|
 i0  0  1  2
  +--+--+--+-     i0= P|i|
  +  3  3  3           |j|
Num-of-cells: 3   Times: 7
(b) array for ProjVec=( 1 0 )

      i j          w  x  y
P=i0|-1 1|, PD=i0|-1 -2  1|
 i0 -2 -1  0  1  2
  +--+--+--+--+--+-
  +  1  2  3  2  1
Num-of-cells: 5   Times: 7
(c) array for ProjVec=( 1 1 )
```

図5 相関関数計算アレーの性能
Fig. 5 Performance of correlation arrays.

（数字の意味は図5と同じ）に示す．これは，Kungにより最初に提案されたシストリックアレーに対応しているが，非同期処理としてみるとセル数が一番多く利点がない．

本システムにより，従来手作業で行ってきたプログラム変換が半自動化され，実現可能な種々の非同期処理アレーを試用することにより，よりよいものを開発する手段が得られた．

5. おわりに

本論文では，非同期並列処理系として非同期処理アレーアーキテクチャを提案し，その仕様記述言語を与

```
for(i=0 to M-1)
 for(j=0 to M-1)
  for(k=0 to M-1)
   { c[i,j]=c[i,j]+a[i,k]*b[k,j];}
(a) matrix product

        a b c
      i|0 1 0|
   D =j|1 0 0|
      k|0 0 1|
(b) data-dependence matrix

ProjVec| cells | times |
 0 0 1 |       |       |
 0 1 0 |  16   |  10   |
 1 0 0 |       |       |
 0 1 1 |       |       |
 1 0 1 |  28   |  10   |
 1 1 0 |       |       |
 1 1 1 |  37   |  10   |
 primi |  64   |  10   |
(c) Number of cells and
    processing times for M=4

       i  j  k          a  b  c
 P=i0|-1  1  0|, PD=i0| 1 -1  0|
   i1|-1 -1  2|      i1|-1 -1  2|
  i1 -6-5-4-3-2-1 0 1 2 3 4 5 6
 i0+--+-+-+-+-+-+-+-+-+-+-+-+-+-    |i0|=P|i|
 -3+          1   1   1   1          |i1|  |j|
 -2+        1   2   2   2   1              |k|
 -1+      1   2   3   3   2   1
  0+  1   2   3   4   3   2   1
  1+    1   2   3   3   2   1
  2+        1   2   2   2   1
  3+          1   1   1   1
 Num-of-cells: 37   Times: 10
(d) array for ProjVec=( 1 1 1 ), M=4
```

図6 行列積計算アレー
Fig. 6 Matrix-Product Arrays.

えた．与えられた問題の解法が標準形プログラムで与えらるとき，それから非同期処理アレーを組織的に設計する方法を与えた．さらに，その作業を支援するものとして構築した設計開発支援システムの概要を説明した．

非同期処理アレーはデータ駆動型であるため，その性能解析は難しく，シミュレーションによる性能評価が必要である．シミュレータの実現によりこのことを可能にした．設計法については，シストリックアレーの設計法の原理を適用して，非同期系に固有の問題を考慮してその解決をはかって考案している．しかし，標準形プログラムとプリミティブアレーの対応が直接的であるため，プログラムにおいて制御変数が減少するループを扱えないなど，解決すべき課題は多い．

4重ループ以上の問題を2次元以下で実現する方法の開発も実際的には重要であるが，今後の課題である．また，詳細を述べることができなかったが，射影された空間の連続性をなるべく保つ方法やプログラムの本体の計算で制御変数に直接依存する場合の実現法などを設計開発支援システムに組み込むことも残された課題である．

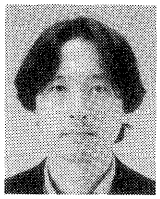
謝辞 本研究を進めるにあたり御討論頂いた北陸先端科学技術大学院大学木村正行教授，東北大学丸岡章教授に深く感謝する．また，本研究の大きな部分は，電気通信普及財団の援助による．ここに記して感謝する．

参考文献

1) Karp, R. M., Miller, R. E. and Winograd, S.: The Organization of Computations for Uniform Recurrence Equations, *J. ACM*, Vol. 14, No. 3, pp.563-590 (1967).
2) Lamport, L.: The Parallel Execution of DO Loops, *Comm. ACM*, Vol. 17, No. 2, pp. 83-93 (1974).
3) Kung, H. T. and Leiserson, C. E.: Systolic array (for VLSI), *Proc. Symp. Sparse Matrix Computations and Their Applications*, pp. 256-282 (1978). (Mead, C. A. and Conway, L. A.: *Introduction to VLSI Systems*, pp. 271-292, Addison-Wesley (1980))
4) Kung, S. Y., Arun, K. S., Gal-Ezer, R. J. and Rao, D. V. B.: Wavefront Array Processors: Language, Architecture and Applications, *IEEE Trans. Comput.*, Vol. C-31, No. 11, pp. 1054-1066 (1982).
5) Moldovan, D. I.: On the Analysis and Synthesis of VLSI Algorithms, *IEEE Trans. Comput.*, Vol. C-31, No. 11, pp. 1121-1126 (1982).
6) Miranker, W.L. and Winkler, A.: Spacetime Representations of Computational Structures, Computing, Vol. 32, No. 2, pp. 93-114 (1984).
7) Rao, S. K. and Kailath, T.: Regular Iterative Algorithms and Their Implementation on Processor Arrays, *Proc. IEEE*, Vol. 76, No. 3, pp. 259-269 (1988).
8) 阿曽弘具：シストリックアレーの自動設計法，電子情報通信学会論文誌 D, Vol. J 71-D, No. 8, pp. 1487-1495 (1988).
9) Lee, P.-Z. and Kedem, Z. M.: Mapping Nested Loop Algorithms into Multidimensional Systolic Arrays, *IEEE Trans. Para. Dist. Sys.*, Vol. 1, No. 1, pp. 64-76 (1990).
10) 阿曽弘具：非同期シストリックアルゴリズムとその設計法，第28回東北大通研シンポジウム論文集「離散アルゴリズム」, pp. 163-172 (1991).
11) 小川誠治，前場隆史，阿部健一：多次元シストリックアレーの系統的設計手法，電子情報通信学会

論文誌D, Vol. J 75-D-I, No. 9, pp. 879-881 (1992).

(平成6年5月12日受付)
(平成7年1月12日採録)



佐藤 健一

1989年東北大学工学部情報卒業. 1991年同大大学院工学研究科修士課程修了. 同年, ソニー株式会社に入社. 以後, 中央研究所にてディジタル信号処理の研究に従事.

阿曽 弘具 (正会員)

1968年東北大学工学部電気卒業. 1973年同大大学院博士課程修了. 同年同大工学部助手. 1979年名古屋大学工学部講師, 同助教授を経て, 1986年東北大学工学部助教授, 現在, 同教授. 工学博士. その間, 学習オートマトン, セル構造オートマトン, 並列処理理論, シストリックアルゴリズム設計論, 文字認識, 音声認識などの研究に従事. 昭和53年度電子通信学会学術奨励賞, 平成3年度電子情報通信学会業績賞受賞, IEEE, ACM, EATCS, 電子情報通信学会, 人工知能学会, LA各会員.