Vol. 36 No. 6　　情　報　処　理　　June 1995

解　説

重点領域研究：超並列原理に基づく情報処理基本体系

3. 超並列プログラミング言語とその処理系†

雨　宮　真　人††

1. はじめに

文部省重点領域研究「超並列原理に基づく情報処理基本体系」において，超並列プログラミング言語に関する研究は主に次の二つの目的を持っている．一つは，超並列計算機システムの可能性と有効性を実用的見地から明らかにしユーザを広く開拓するために，実用的な超並列言語およびそのプロトタイプを開発することである．もう一つは，超並列計算機システムのあるべき姿を求めてその新原理を追究するために既存のプログラミング言語にとらわれない新原理に基づいたプログラミング言語を開発してその可能性を探ることである．

第1の目的に沿った言語は，特に実用性を重んじ，また，多くのユーザを開拓するために広く使用されているC言語をベースとして開発している．この言語はNCXと呼ばれる．NCXはとりあえず並列処理効果（特にデータ並列の効果）が顕著と思われる科学技術計算や画像処理などの分野を適用領域として想定している．第2の目的に沿った言語は並列処理の新しいパラダイムを研究するための実験言語として開発している．この言語は超並列**V**言語と呼ばれる．**V**言語はもう少し広い領域を対象として考え，データ並列とオブジェクト並列の統合的記述を目標としている．

将来はこの二つの研究が融合し，新方式の実用的プログラミング言語を開発するということを目指しているが，現在は，まずそれぞれの目的に沿った言語を開発するということで研究を進めている．

本稿では，はじめに超並列記述の視点と言語設計における問題点を整理し，これらの問題を考慮しつつ設計した二つの言語，NCXと**V**言語についてそれぞれその概略を紹介することにする．

2. 超並列記述の視点と問題点

超並列処理はどのような概念で記述するのが自然であろうか．並列処理とは基本的には多数のプロセス（仮想的なプロセッサと思ってよい）が個々にデータの処理を行うということであるが，これらが完全に個々独立に行うことはなく必ずプロセスの間でデータの交換（通信）や処理の同期が必要となる．並列処理記述の問題は，結局，個々のプロセス内部の処理とプロセス間の通信と同期をどのように統一的に記述するのが自然かという問題に帰着される．

（1）　並列記述の意味論

並列処理言語の意味論は，まず，同期の観点から見ると，同期的実行意味論（以下SIMD（Single-Instruction Stream Multiple-Data Stream）と呼ぶ）と非同期的実行意味論（以下MIMD（Multiple-Instruction Stream Multiple-Data Stream）と呼ぶ）に大別される*．また，何を対象として並列処理の制御を記述するかにより，データ並列とオブジェクト並列に分けられる．データ並列ではアレイやベクトルなどのような多数のデータに同一の演算を同時に施すという視点で処理を記述し，そこにはプロセスの間の同期制御は陽に現れない．意味論的にはSIMDと相性がよ

† Research on Massively Parallel Languages and Their Implementation by Makoto AMAMIYA (Interdisciplinary Graduate School of Engineering Sciences, Kyushu University).

†† 九州大学大学院総合理工学研究科

* ここでの意味論とはその言語の計算モデル，つまりその言語を実行する抽象並列マシンのモデルを指す．抽象並列マシンのモデルにはPRAM（Parallel Random Access Machine）モデルなどがよく用いられる．また，SIMD, MIMDという語は一般には並列マシンのアーキテクチャのモデルとして用いられている．しかし，ここでは意味論をプロセッサ間の同期や通信など，より現実を反映させた抽象並列マシンのモデルでとらえるのが自然であると考えられるのでSIMD, MIMDという語を用いることにする．

い．これに対してオブジェクト並列では個々のオブジェクトの間の通信を契機にオブジェクトの実行・同期を記述する．並列記述の視点はオブジェクト（データおよびプロセス）にあり，その意味論はMIMDと相性がよい．（並行プロセスの記述はオブジェクト並列記述の良い例である．）

並列処理の意味論に関して，SIMDをベースにするのが自然であるかMIMDをベースにするのが自然であるかは議論のあるところである．SIMD意味論は並列効果が直観的で分かりやすい．しかし，命令レベルでの同期実行では，処理の自由度を得ようとすれば，休止状態のプロセッサが多くなり，効率は低下する．このため記述対象は限定されたものとなる．（個々のプロセッサが並列に実行できる処理の区間を並列処理単位といい，並列処理単位の大きさを粒度という．SIMDは粒度が非常に低い．）

一方，MIMD意味論では処理記述の自由度はあるものの，命令レベルの完全非同期実行では随所に現れる同期と通信のためにオーバヘッドが大きくなり現実的でない（命令レベルのMIMDもやはり粒度が非常に低い）．そこで，SIMDとMIMDの中間で同期実行の制御を行うのが現実的ということになる．これはSPMD（Single-Program Multiple-Data Stream）と呼ばれている．SPMDはSIMDの立場で見れば同期実行の単位が命令列（プログラム）レベルになり，MIMDの立場で見れば同期の契機が命令列（これをスレッドということもある）レベルになる．ともに粒度が高くなる．

計算機のハードウェアはできるだけ汎用性を持ったものとして開発するのが普通であるから，JUMP-1[1]をはじめとして多くの超並列計算機のハードウェアはMIMDタイプである．さらに，これらのハードウェアにはプロセッサの間の処理の同期をとるための制御機構が備わっている．これは一般にバリア同期機構と呼ばれる．バリア同期では，指定された同期点でプロセッサ間の同期がとられるが，同期点の間では個々のプロセッサはそれぞれ非同期に実行できる．バリア同期の区間をできるだけ大きくとり粒度を上げることができればそれだけ個々のプロセッサが非同期に独立して実行できることになり同期や通信のためのオーバーヘッドを減らすことができる．バリア同期機構を備えた計算機を効率的に動作させるという観点からは，SIMD意味論あるいはMIMD意味論で記述されたプログラムからできるだけ少ないバリア同期点を抽出するということが言語処理系に課せられた重要な問題となる．

（2） データ並列とオブジェクト並列

データ並列の構造はSIMD型の計算に対応する．つまり，同一演算（あるいはプログラム）を多数の異なるデータに適用することによって得られる効果であり，データのサイズに比例する並列効果が得られる．データ並列のプログラムはSIMD型の並列計算機を想定すれば比較的記述が容易である．しかし，処理対象はベクトル，行列，メッシュ，集合などの単純な規則性を持った構造（これを整構造という）のデータに対して，それらの各要素に同一処理を施すようなアルゴリズムに制限される．現在，顕著な並列処理効果が期待されているのは科学技術計算や画像処理の分野であるが，それはもっぱらこのデータ並列によるものである．

商用超並列計算機用のプログラミング言語としてC言語をベースとしたData Parallel C[2]やC*[3]，Fortranをベースにした HPF[4]などがあるが，これらはみな基本的にはデータ並列の記述を念頭に置いたものである．

一方，オブジェクト並列はMIMD型の計算に対応する．オブジェクト並列はMIMD計算という意味でより広い領域をカバーできる．（粒度の高いレベルでは，オブジェクト並列はオブジェクト間の相互作用をメッセージ通信によって記述し，並列実行の制御はメッセージ通信の枠内で記述する．）しかし，一般に演算（あるいはプログラム）の間にはデータや制御の依存関係があるため必ずしもデータのサイズに比例した並列効果が得られるとは限らない．また，プログラム記述の観点からは，ユーザが直接オブジェクト並列の制御を意識してプログラム記述することはそれほど容易ではなく，ユーザは書いたプログラム中に潜む並列性を自動的に抽出してくれるような言語処理系を期待したくなる．プログラムに潜む並列性を抽出し（通信や同期のオーバヘッドの問題が克服できて）効率的に実行できるようなハードウェアが開発できればその効果は大きいが，一方，言語処理系によって，バリア同期点を抽出し粒度の

高い並列制御を行う方式の開発もまた重要である．粒度を高くすればオブジェクト・レベルの並列性が期待でき，低くすればさらにスレッド・レベルの並列効果が期待できる．現在のところ実用的な意味でオブジェクト並列が顕著な応用領域はまだ明確でなく，今後の研究に待つところが大きい．オブジェクト並列の代表的な例はActorモデル[5]であり，Actorモデルに基づいて実際に開発された言語にABCL[6]がある．

3. 実用的な超並列プログラミング言語NCX

本言語は実用性を重視し，データ並列に焦点をあてて言語仕様の設計を行った．以下，NCX言語設計の考え方および言語の特徴についてその概略を紹介する．NCXのより詳細な解説は文献7)～9)を参照されたい．

（1） 実行モデル

NCXの実行モデルはSIMD意味論が基本となっている．ただし，MIMD実行が自然だと考えるユーザにはSIMDモデルと同一のセマンティクスでMIMD実行を保証している．(たとえば，条件文や繰返し文などの実行では真部実行と偽部実行を並列に行いその出口で必ず同期がとられると考えても，真部実行と偽部実行を逐次的に行うと考えてもよい．ユーザ言語レベルではどちらも意味は同じである．また，繰返し文の実行や関数実行などにおいては，並列に実行する繰返し文や関数間で互いにデータの参照・交換を行うような関数ではその入口と出口で同期がとられると考える．)

SIMDベースの実行ではユーザは仮想マシンとしてSIMDマシンを前提とし，処理をすべてSIMDイメージで記述する．しかし，実行マシンはSIMDマシンに限定しない．MIMD（あるいはSPMD）マシン上での実行も最大限のマシン性能を引き出せるよう，言語処理系でターゲットマシンの機能・構造を活かしたMIMD（あるいはSPMD）実行コードを生成する．マシン実行の際には演算ごとの同期をとることをせず必要最小限のポイントでのみバリア同期をとる．この必要最小限の同期ポイントの抽出は言語処理系が同期点解析によって行う．

（2） データ構造とデータマッピング

NCXではフィールドの概念を導入する．フィールドは仮想プロセッサの集合であり，並列処理の対象となるデータ集合と1対1対応にプロセッサ集合が割り当てられる．フィールドの定義の際にプロセッサ間結合のトポロジーを与えることができる．フィールドの基本トポロジーにはベースフィールドとしてmesh, binarytree, hypercubeが用意されている*．

データ構造の変化に対応してフィールドを切り替えたりフィールドの次元を上げたりすることができ（たとえば次元を上げるためにはspawn文が用意されている），異なるフィールド間で変数の代入を行えばそのときデータの転送が引き起こされると考える．データの処理はフィールド内の1プロセッサ要素に対して処理を記述する．同一フィールド内の他の要素データを参照するにはフィールドにインデックスを持ち込みこれによって特定する．

（3） 変数定義および参照

データ並列の対象となる変数はフィールド名をともなって宣言され，同じ名前の変数が指定されたフィールドのすべての仮想プロセッサに割り当てられる．変数の参照には変数名に加えて，フィールドインデックスを用いてそれがどの仮想プロセッサに割り当てられたものであるかを特定する．

（4） データ並列計算のビュー

データ並列の計算は現在実行中のフィールド（これをカレントフィールドという）内の一つの仮想プロセッサに視点をおいて記述する．同じ計算が同じフィールドのすべてのプロセッサ上で同期して実行される．代入式の実行ではアクティブなすべての仮想プロセッサが同時に右辺を評価してその値を求め，すべての仮想プロセッサがその値を求め終わったときに，代入が行われる．ベクトル要素の総和や積，最大（最小）値，などを求めるリダクション演算子は基本演算子として用意されている．

（5） 関　数

関数の呼出しは，アクティブなすべての仮想プロセッサが同時に行う．同時に呼び出される関数とその実引数の個数はアクティブなすべての仮想

* このほかに，仮想プロセッサ1個のみからなる特殊な組込みのフィールドmonoが用意されている．monoフィールド内では仮想プロセッサが1個であるから，プログラムの実行は逐次的となる．

プロセッサで同じでなければならないが，その実引数の値は異なっていてよい．関数には，(a)任意のフィールドから呼び出すことのできる関数，(b)特定のフィールドのみから呼び出される関数，(c)ベースフィールド上のフィールドのみから呼び出すことのできる関数を区別して定義することができ，呼出しに制限を与えることができる．関数への引数にはフィールド情報を付加することができ，これによって関数本体内でのフィールドデータへのアクセスを制御することができる．

（6） 入 出 力

C言語が提供する逐次型の基本入出力（monoフィールド内で実行される）に加えて，フィールド内（並列実行部分）から直接入出力を行わせる並列入出力インタフェースを与えている．並列入出力インタフェースには，各仮想プロセッサがそれぞれ直接データにアクセスできる並列ストリームと，フィールド中の各仮想プロセッサが自分の入出力の対象となるファイルを自分の仮想空間にマップすることのできる並列メモリマップトファイルとを用意する．これら並列入出力の仕様についてより詳しくは文献10）を参照されたい．

（7） プログラム例

次にNCXプログラムの簡単な例を示す．このプログラムはN×Nの行列の積を計算するものである．このプログラムは，ベースフィールドmesh上にN×Nの2次元フィールドmatrixを宣言し，この中で行列の乗算を行っている．a, b, cはmatrix上の変数であり，aとbの行列積がcに求められる．2次元フィールド上の仮想プロセッサP (i, j) がc (i, j) を計算するが，このときspawn (k : N) によって次元が拡張され，N×N×Nの3次元フィールド (i, j, k) 上で仮想プロセッサ {P (i, j, k) | 0≦k<N} が計算に参加し，内積c@(i, j) +=a@(i, k) *b@(k, j) を求める．+=はベクトル和を求めるリダクション演算子である．

```
field matrix (N, N) on mesh;
int a, b, c on matrix;
in matrix (i, j) {
  c=0;
  spawn (k : N)
    c@(i, j) +=a@(i, k) * b@(k, j);
}
```

また，次の例は上の行列積計算のプログラムを特定のフィールドから呼び出される関数として定義したものである．この関数は

```
field a1 (10, 10) on mesh;
```

と宣言されているフィールドa1のみから呼び出されるa1専用の関数である．

```
int matmul (a, b, c) in a1 (i, j)
int a, b;
{
  int c=0;
  spawn (k : 10)
    c@(i, j) +=a@(i, k) * b@(k, j);
  return (c);
}
```

4. 実験的超並列プログラミング言語V

超並列プログラミング言語VはNCXとは対極の位置にある実験言語である．NCXが実用性を強く意識した言語であるのに対し，Vは自然なMIMDセマンティクスの追求，粒度を意識しないオブジェクト並列およびデータ並列の記述を実験・検証するための言語である．

以下，Vの設計思想，特徴の概略を紹介する．Vの言語仕様について詳しくは文献11），12）を参照されたい．

（1） 実行モデル

Vのベースとなる MIMD 意味論では，基本的にはメモリ書き換えおよび実行順序制御の概念はない．ユーザはただデータの定義，参照関係のみを意識して演算レベルの非同期実行をイメージする．同期や実行順序スケジュールはデータ依存則にのっとって自動的に制御されていると考えてプログラム記述を行う．並列実行スレッドの抽出，並列分岐・併合 (fork-join) などの同期点の抽出，スレッドの最適配置などは言語処理系が行い，マシン実行の段階ではできるだけ同期のオーバヘッドを減らすようにする．

本言語は関数型（データフロー）言語の意味論をベースとするが，内部状態を持ったオブジェクト（Vではagentと呼んでいる）の記述を許すなど必ずしも純関数性にはこだわらない．メッセージ/データ通信と演算とのインタフェースはデータ依存則の枠組で統一される．

（2） データ並列

データ並列の記述では，ベクトルやアレイの要素ごとの計算は，同一演算（関数）を全データに施す高階関数 Apply-all を基にした foreach 式によって記述する．foreach 式は非同期の fork-join 展開によって並列実行される．アレイやベクトルなどのデータ構造はその構造が完成していなくても構造の要素に直接アクセスすることを許すノンストリクトなものとし，データ並列の効果をできるだけ抽出できるようにする．構造データの基本は書き換え不可（immutable 型）のものであるが，ユーザに読み書きの同期管理の責任のもとで書き換えを許す mutatble 型も導入する．

（3） オブジェクト並列

履歴や内部状態を持った処理体を agent と呼び，agent の定義と agent 間のメッセージ/データ通信の記述手段を与える．agent は内部状態を持ったオブジェクトであり，互いにメッセージを介して通信・同期を行い並列に処理を実行する．また，並列オブジェクトの集合体を定義する agent 集合の定義機能を与え，規則的な構造を持った並列オブジェクト集合に対しては特に効率的な実行を可能とする．このために，agent の規則的な配置を意識した agent 配列の概念を導入する．agent の実行は非同期である．

メッセージ/データの受信点で自動的に待ち状態になり，メッセージ/データの到着により自動的に待ちが解かれる（メッセージ駆動）．また，パターン照合によるメッセージの同定・処理起動に加えて，効率的なメッセージ/データ通信を実現するためにポート（import, export）の概念を導入し，agent の特定受信点に直接メッセージ/データを転送する機能を与える．

（4） プログラム例

NCX との対比として行列積を計算する関数の例を以下に示す．プログラム中，たとえば foreach (i, j) in ([1..n], [1..n]) body Exp は n×n 個の各 i, j について Exp を並列に実行する．また，sum-of は組込みのリダクション関数である．

```
function matmul (a, b : matrix) return matrix
= foreach (i, j : integer) in ([1..n], [1..n])
    body sum-of (foreach (k : integer) in[1..n]
      body a[i, k]*b[k, j])
```

また，次の例はポートを介した agent 間通信の記述例である．このプログラムは input で与えられる整数ストリーム要素中の 2 番目に大きな値を output とする．join オペレータは生成された agent 活性体 max1 と max2 をポート line を介して結合する．max1 からの出力は line を通して max 2 の入力となっている．プログラム中 for (x) init (v) body... recur (v′) は再帰式と呼び，初期値を v 再帰パラメータを v′として末尾再帰の実行（つまりループ実行）を行う．また，put (x) after put (y)は，put (y) を行った後に put (x) を行うことを指定する（**V** では必要な場合には逐次実行を明示的に指定する．）

```
agent selectmax ()
  {import numbers : integer}
  {export max, others : integer}
= for (tmp : integer, nums : stream)
init
    (head (numbers), tail (numbers))
    body
  if numbers=nil then put (max, cons (tmp, nil))
    after put (others, nil)
  elsif tmp<head (nums)
    then recur (head (nums), tail (nums))
    after put (others, tmp)
  else recur (tmp, tail (nums))
    after put (others, head (nums)) ;

= {join max1=create (selectmax) {input} {_, line},
    max2=create (selectmax) {line} {output, _} ;
```

5. 処 理 系

NCX の処理系構成手順の概略を**図-1** に示す．第1フェーズで，SIMD 型のソースコードを MIMD 実行モデルを持った仮想マシンのコード（中間言語）に変換する．この中間言語は C 言語＋同期プリミティブ（barrier 同期点挿入），仮

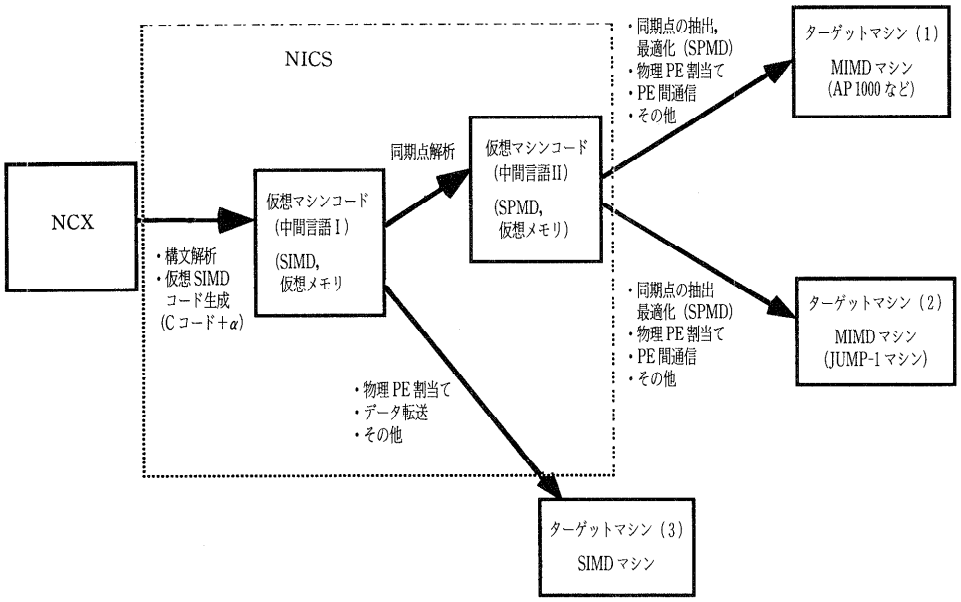

図-1 NCX 言語処理系

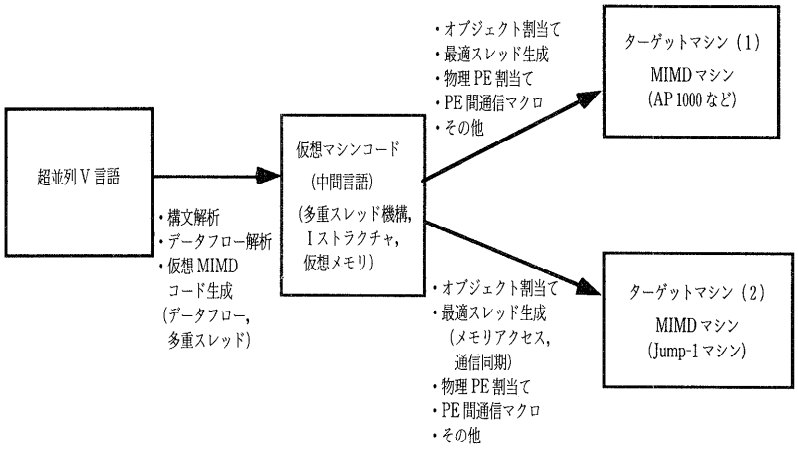

図-2 V 言語処理系

想プロセッサを想定したエミュレーションループ構造の MIMD マシン中間コードで構成される．次のフェーズで実マシンコードに変換する．このフェーズでは実マシンの構造・機能に応じた最適コード（SPMD コード）を生成する．特に，実マシンにおける同期制御コードへの変換，実マシン上での仮想プロセッサ多重化，(多重) スレッド化，スレッド切替え制御，実プロセッサ間通信制御などの命令コード（マクロ）の埋め込み・生成を行う．処理系は NICS (NCX Intermediate Code Synthesizer)[13]と呼ばれるプリプロセッサを使用し，NCX プログラムをターゲットマシンの C 言語に変換する．

超並列 V 言語の処理系構成手順の概略を図-2 に示す．超並列 V 言語の中間コードは仮想マシンとして多重スレッド高速処理演算部と通信制御部，メモリアクセス制御部を持ったプロセッサの結合体を想定する．この仮想マシンは細粒度のスプリットフェーズ型多重スレッド実行機構を持ったものを前提としている．

6. おわりに

本研究は，文部省重点領域研究「超並列原理に基づく情報処理基本体系」の中で「超並列記述系・処理系研究班」を組織して行った．

研究班は以下の11人のメンバで構成されている．

雨宮真人（九州大学大学院総合理工学研究科），佐藤雅彦（東北大学電気通信研究所），牧之内顕文（九州大学工学部），萩原兼一（大阪大学基礎工学部），湯浅太一（豊橋技術科学大学情報工学系），相田仁（東京大学工学部），上田和紀（早稲田大学理工学部），井田哲雄（筑波大学電子情報工学系），荒木啓二郎（奈良先端科学技術大学院大学情報科学研究科），馬場敬信（宇都宮大学工学部），西川博昭（筑波大学電子情報工学系）．

NCXの言語仕様は湯浅太一教授を中心とする豊橋技術科学大学のグループがその骨格を設計し，本研究班全員がほぼ3カ月に1回程度の研究会を開いて議論し仕様の詳細化を行った．NCXの言語処理系は，現在，富士通AP 1000, SUNワークステーション，など種々の計算機をターゲットマシンとして実装中である．また近い将来JUMP-1への実装も計画している．NCX処理系はNICSを用いて開発できるので多くのマシンへの移植も比較的容易である．

V言語の仕様については九州大学のグループが設計し，本研究班で議論した．**V**言語の処理系は現在富士通AP 1000上への実装を行っている．**V**言語についても近い将来JUMP-1への実装を行う予定である．

参考文献

1) 富田眞治：超並列計算機JUMP-1のアーキテクチャ，情報処理，Vol.36, No.6, pp. 528-537 (1996).
2) Hatcher, P.J. and Quinn, M.J. : Data-Parallel Programming on MIMD Computers, The MIT Press (1991).
3) Thinking Machine Corporation : C*Programming Guide, Version 6.0 Beta (1990).
4) High Performance Fortran Forum : High Performance Fortran Language Specification, 1993. Available from titan.cs.rice.edu by anonymous ftp.
5) Agha, G. : Actors : A Model of Concurrent Computation in Distributed Systems, The MIT Press (1986).
6) Yonezawa, A., Matsuoka, S., Yasugi, M. and Taura, K. : Implementing Concurrent Object-Oriented Language on Multicomputers, IEEE Parallel and Distributed Technology, pp.49-61 (May 1993).
7) 湯浅，貴島，小西：データ並列のための拡張C言語NCX，電子情報通信学会論文誌，D-1 Vol. J 78-D-1, No.2, pp.200-209 (Feb. 1995).
8) 超並列C言語NCX言語仕様書 (Version 3)，文部省重点領域研究「超並列原理に基づく情報処理基本体系」第4回シンポジウム予稿集，pp. 2.8-2.92 (Mar. 1994).
9) Yuasa, T., Kijima, T. and Konishi, Y. : The Data-Parallel C Language NCX and Its Implementation Strategies, in Theory and Practice of Parallel Programming, Springer LNCS 907, pp.433-456 (Apr. 1995) (to appear).
10) 天野，牧之内：超並列C言語NCXの超並列入出力インターフェイス，文部省重点領域研究「超並列原理に基づく情報処理基本体系」第4回シンポジウム予稿集，pp.2.8-2.92 (Mar. 1994).
11) Kusakabe, S. and Amamiya, M. : A Dataflow Language with object-based Extension and Its Implementation on a Commercially Available Parallel Machine, Proceedings of the ACM International Conference on Supercomputing 1995 (July 1995) (to appear).
12) 日下部，雨宮：超並列超並列V言語，文部省重点領域研究「超並列原理に基づく情報処理基本体系」第4回シンポジウム予稿集，pp.2.143-2.190 (Mar. 1994).
13) 湯浅：NICS : NCX Intermediate Code Synthesizer, 文部省重点領域研究「超並列原理に基づく情報処理基本体系」第4回シンポジウム予稿集，pp.2.93-2.111 (Mar. 1994).

(平成7年3月13日受付)

雨宮 真人（正会員）

昭和42年九州大学工学部電子工学科卒業．昭和44年同大学院工学研究科修士課程修了．同年日本電信電話公社(現在NTT)武蔵野電気通信研究所入所．以来プログラミング言語・処理系，自然言語処理，関数型/論理型，知能処理アーキテクチャ，などの研究に従事．現在九州大学大学院総合理工学研究科教授．工学博士．著書「コンピュータ・アーキテクチャ」(オーム社) 他．電子情報通信学会，日本ソフトウェア科学会，人工知能学会，IEEE，ACM各会員．