計算機アーキテクチャ 89−26
（１９９１．７．１９）

『発注／受領型』並列計算機上の並列手続き型言語・実行系・応用例

星野 浩志　冨澤 眞樹　五十嵐 智　阿刀田 央一

東京農工大学工学部

我々は、高速なプロセス生成と実行管理を命令レベルで持つ分散共有メモリ型並列計算機を提案している。C//は、この並列計算機の基底言語として設計された並列手続き型言語である。本言語は、関数呼出しによって動的に生成されるプロセス群が作り出す拡散的な制御フローと、スコープ規則と存続期間および引数の受け渡しから導かれる変数の多重性と階層的な共有性とによって並列プログラムを記述する。本稿では、C//を提示し、そのコンパイル技法、実行時ルーチン、およびプロセッサへの関数割り当て方法について報告する。また、C//の実際問題の応用例として、画像解析による３次元計測のアルゴリズムをC//で記述した経験について報告する。

Design, Implementation and Experience of Parallel Procedural Language on Demand/Accept Multiprocessor

Hiroshi Hoshino　Masaki Tomisawa　Satoshi Igarashi　Oichi Atoda

Faculty of Technology, Tokyo University of Agriculture and Technology
2-24-16 Naka-machi, Koganei-shi, Tokyo 184, Japan

We have proposed a distributed control mechanism for multi-processor, based on parallel procedure call with invoking processes dynamically. In this report, we present parallel procedural language C// executing on this mechanism. C// provides a variety of multiplicity and uniqueness of control flow and data, which allows to describe logical parallelism. The scope-rules and storage-classes in C// as well as implementation of compiler and environment for execution are discussed.

3．言語の持つ多重性の枠組み

3．1　transfer変数の位置づけ

本言語では、transfer変数を用いて制御に事実上の名前づけを行う。関数インスタンスの識別子をそのまま値として扱うことはしない。すなわちtransfer変数は、プログラマが可視の名前によって、前倒し呼びや受取りを積極的に管理しようという意図をもつ。

transfer変数は、2．で述べたように、制御を意味し、宣言やスコープ及び存続期間は自動変数と同様である。しかしtransfer変数は、構文的には変数としてふるまう。このようにtransfer変数の言語上の性格づけには、制御とデータの間での選択の幅があり、その結果、言語の性質は大きく違ったものになり得る。たとえばtransfer変数に外部的なスコープを許すと、制御フローが分岐した親以外のフローに合流できることになる。また関数の引き数や返り値になることを許すと、制御フローの動的なつなぎ替えができることになる。これらは興味深い問題を含むが、関数インスタンスが自分の呼出し元を消滅させてしまうような、一貫性管理やデバッグに厄介な可能性を持ち込んでしまう。従って現段階では、transfer変数に対する並列プリミティブ以外の演算や、関数の引き数および返り値となることを禁止する。

3．2　変数の多重化と共有

自動変数は関数インスタンスの生成に伴って多重化され、インスタンスで局所的な変数となる。外部的な変数は、インスタンス間で共有される。このように、逐次型言語のスコープを直ちにそのまま多重性の規則に拡張できる。一方、内部静的変数は、C言語と同様に多重化しないで静的領域に割り当てるか、自動変数と同様に多重化するかの選択がある。本言語では、内部静的変数を、前倒し呼びを越えた関数の状態を保持するものと考え、同一関数から生成された関数インスタンス間で共有されるとした。

3．3　引き数の渡し方と多重性／共有性

複数の関数インスタンスに同じ引き数を値渡ししたとき、それらはそれぞれの関数インスタンス内で領域が与えられるので、すべて独立となる。しかしアドレスで渡されたときは、複数のインスタンスで共有される。たとえばある関数インスタンスの自動変数のアドレスが、そのインスタンスの呼び出した関数インスタンス群に渡されると、その自動変数は呼び側と呼ばれ側の関数インスタンス群で、共有されることになる。このように、本言語では共有は階層性をもち、共有させるかどうかの指定は呼び側が行う。

一方、値渡しと参照渡しとの中間に位置する引き数の形式が考えられるので、言語仕様に含めた。前倒し呼びに伴って、

```
t //= func(x$);
```

と記述すれば、xは一旦値渡しされるが、関数インスタンス内での変更が //tに伴って x に遡及する。x は左辺値とし、$ は『遅延代入演算子』と呼ぶことにする。

3．4　排他・同期

本言語では、共有変数に対する排他や同期は、lock文によって記述する。複雑な同期機構の必要性は問題依存なので、言語側に含ませず、lock文を核にマクロ等で用意する。lock文は、次のように記述する。

```
lock(g;h) { … }
```

{ … }内の区間に入るとき g をロックし、出るとき h をアンロックする。g、hは keyholeという型をもった変数である。この変数のスコープ規則は、一般の変数と同じである。

3．5　不確定選択

本言語では関数インスタンスの OR 型選択のプリミティブとして、among文を含ませた。among文が実行されたとき、関数インスタンスのうちのリターン済みのものの中から一つ選択される。among 文は、transfer変数を用いて次のように書く。

```
among(i; i < M; t[i]); /*OR 型選択*/
y = //t[i];            /*受取り    */
```

この例では、among文は transfer変数 t[i]（int i, 0≦i＜M）の内の一つを選択し、このときの制御変数iの値を保存して次の文に移る。返り値はあらためて //演算子で得る。among文が実行された時点で、もしすべてがまだ実行中なら、どれかがリターンするまで自動的に待ち、結果的に時間依存の選択がなされる。

4．C//実行系の考え方

C//の実行系に対して要求されることは、次の6点に整理できる。①軽負荷で、動的にインスタンスを生成できる。②一つのプロセッサで、複数インスタンスを並行実行できる。③並行実行インスタンス同士が飢餓状態に陥るのをふせぐことができる。④任意のプロセッサで、任意の関数のインスタンスを生成できる。⑤一つの関数コードを、複数のプロセッサにコピーして持たせることができる。⑥頻繁に参照するデータは、ホームメモリに置くことができる。

たとえば図1の例では、小規模な関数が再帰的に前倒し呼びを行い、多数のインスタンスが生成される。このとき①が要求される。またC//では、計算機の数

などの物理的構成を記述しないので、関数インスタンスの個数がプロセッサの数を越えたときの処理が記述できない。そのため実行時に②が要求される。さらに図1では、一つの関数から生成された複数のインスタンスが並列に実行されることを想定している。そのため④が要求される。このとき当然⑤も要求される。複数のプロセッサ要素にコピーされた関数コードを、『関数クローン』と呼ぶ。以上をまとめると、C//の実行状態のモデルは図2のようになる。

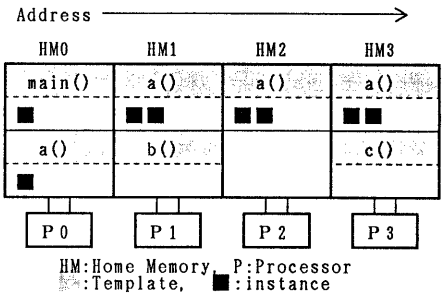


図2　C//実行状態のモデル

6点の内、①②③はOSのプロセス管理で一応実現できる。しかし多数のインスタンスの動的生成が要求されるとき、集中型のOSによるシステムコール等では性能を維持できない。そこで5．で述べる分散型並列制御機構[1]で、プロセッサ要素に機械語レベルの並列制御命令をもたせて実現した。コンパイラは、C//の並列プリミティブを、対応する並列制御命令に展開するだけでよい。6．では、コード展開の指針を述べる。

④⑤⑥は、論理的な関数インスタンス、関数コード、データを、どのようにして物理的なプロセッサに分配するかという問題である。関数インスタンスの分配は実行時ライブラリで行い、関数クローンとデータの分配はローダで行う。これらは7．で述べる。8．では、実行時ライブラリについて述べる。

実行系は、コンパイル、リンク、ロードという一般的な手順で実行状態を作る。コンパイル、リンクでは、すべての関数コードを、再配置可能形式で一つのオブジェクトファイルにまとめる。ローダはロード条件ファイルにしたがって、関数コードを分配する。これらの処理は並列計算機のプロセッサエレメントの内の一つで行われる。ローダはコードをロードした後、自分自身のプロセッサを含めて並列制御機構を立ち上げる。このときローダを実行していたプロセッサは、カーネルプロセッサ（KPE）と呼ばれる、特権的なプロセッサになる[4]。カーネルプロセッサがmain()を発注して、実行が開始する。

5．　分散型並列制御機構

5.1　制御機構の考え方

『発注／受領型』並列計算機の分散型並列制御機構は、次の方針で設計された。①インスタンスの動的生成、実行管理はプロセッサ要素に分散してもたせる。②プロセッサ要素に機械語レベルの並列制御命令（表1）をもたせ、高速化を図る。③制御機構に必要なデータをホームメモリに置く。④インスタンス間の引き数受渡し機構を制御機構にもたせる。

②の並列制御命令として、dmnd 命令、acpt 命令、rsod 命令を用意した。これらはC//における前倒し呼び、受取り、関数からのリターンに対応するものである。それぞれを会社間の仕事の発注、受注に対応させて、「発注」「受領」「納品」と呼ぶ。

5.2　分散型並列制御機構

並列制御機構を実現するため、各プロセッサエレメントにプロセッサヤードと呼ぶ制御構造（図3）を分散させて配置する。最初プロセッサは、実行すべきインスタンスを持たないので、デマンドキュー（Demand Queue:ＤＱ）に要求が積まれるのを待つ。新しい要求が積まれると、それにICB（Instance Control Block）とIDB（Instance Data Block）を割り当て、要求で指定された手続きの実行を開始する。制御機構では手続きの単位をテンプレートと呼ぶ。テンプレートの実行開始番地はテンプレートベクタ表（Template Vector Table）から知る。

dmnd 命令は、自分のICB内に受信点と呼ぶ構造（図4）を確保し、受注側のＤＱに要求として受信点アドレスを積む。受信点は、受注側インスタンスを識別するための構造で、dmnd命令の二つの入力オペランドを持つ。受信点は dmnd命令ごとに確保され、そのアドレスが返される。acpt命令は、与えられた受信点アド

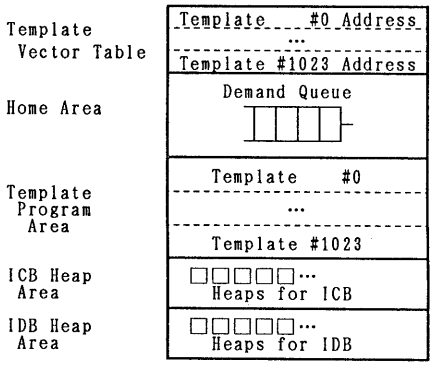


図3　プロセッサヤードの構造

レスから、受け取るインスタンスを知る。acpt命令は、受信点のRフラグから受注側インスタンスが終了したかどうかを知る。終了していれば実行を続ける。終了待ちならば別のインスタンスを実行するため、ＤＱの走査に戻る。

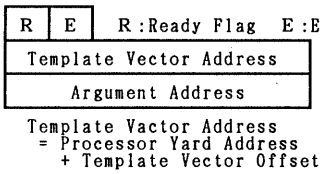


図４　受信点（Rcp.:Recipient）の構造

６．制御命令とコンパイルの指針

我々は分散型並列制御機構を、プロセッサエレメントにMC68010を使った分散共有型並列計算機上で、エミュレーション命令によって実現した。

Ｃ//が持つ並列記述のプリミティブは、ほとんど制御機構に組み込まれている。コンパイラはＣ//を、これらの命令に展開すればよい。制御命令、制御構造とＣ//の対応関係の一覧を、表２(a),(b) に示す。

６．１　並列プリミティブのコード展開

前倒し呼びは、dmnd命令に展開する。dmnd命令は、インスタンス識別値として受信点アドレスを出力する。Ｃ//実行時には、transfer変数は、内部的にこの受信点アドレスを値として持つ。transfer変数に対する演算子 //, /! も、受信点アドレスを入力オペランドに持つacpt, rtct命令に展開できる。また among, lock 文も、インスタンス切り替えを伴った同期命令 slct, tass に展開できる。遅延代入は、引き数領域に特別な構造をもたせて、コード展開で解決する。

６．２　変数の確保

自動変数はインスタンスごとに多重化される。制御機構ではスタックをIDBにとるので、自動変数は従来Ｃのコード展開で自動的に多重化される。一方外部変数と、関数クローン間で共有される可能性のある静的変数は、コードとは切り放して参照関係のリンク、ロードを行う。

６．３　関数のコード展開

Ｃ//の関数内部のコード展開は、並列関係のプリミティブを除いて、従来のＣとほぼ同じと考えてよい。ただし順序呼出しと return 文のコード展開が異なる。

同じ関数に対して前倒し呼び、順序呼びの混用を許すので、関数のコードはどちらで呼ばれたときも、同じように実行されるものでなければならない。このとき問題になるのは、引き数リンクとリターンの処理である。引き数リンクは、dmnd命令と同じ引き数リンクを持った jsra命令で解決できる（６．４参照）。一方リターンは状態によって、納品または手続きからのリターンをする rsod (return subroutine or deliver)

表２(a)　制御命令と言語の対応関係

動作	制御命令	入力ｵﾍﾟﾗﾝﾄﾞ	出力オペランド	Ｃ//との対応
発注	dmnd 命令	ﾃﾝﾌﾟﾚｰﾄﾍﾞｸﾀｱﾄﾞﾚｽ と引き数ｱﾄﾞﾚｽ	受信点ｱﾄﾞﾚｽ	t //= func()
受領	acpt 命令	受信点ｱﾄﾞﾚｽ	引き数アドレス	//t
納品または 手続きからのﾘﾀｰﾝ	rsod 命令	0 または 帰り番地		return
取消	rtct 命令	受信点アドレス		/!t
不確定選択	slct 命令	受信点ｱﾄﾞﾚｽﾘｽﾄ のｱﾄﾞﾚｽと個数	ﾘｽﾄ中の番号	among 文
相互排他	tass 命令	tas ｱﾄﾞﾚｽﾘｽﾄ のｱﾄﾞﾚｽと個数		lock 文
手続き呼出し	jsra 命令	手続きのｱﾄﾞﾚｽ と引き数ｱﾄﾞﾚｽ		func()

表２(b)　制御構造と言語の対応関係

制御モデル	制御構造	Ｃ//との対応
ｲﾝｽﾀﾝｽ識別値	受信点ｱﾄﾞﾚｽ	transfer変数
手続き	テンプレート	関数

命令で解決できる。rsod命令は、スタックに帰り番地が積まれているときはその番地にリターンし、0 が積まれているときは、インスタンスを終了する。この 0 は、制御機構側でインスタンス起動時にセットする。

6.4 引き数受渡しのコード展開

(1) 引き数受渡し機構

dmnd および jsra 命令には、二つの専用レジスタ AP (Argument Pointer) と PP (Parameter Pointer) を使って自動的に引き数領域のアドレスを受け渡す機構が、組み込まれている。呼び側では AP に引き数領域のアドレスをセットして dmnd または jsra 命令を実行する。呼ばれ側の実行が開始したときには、APの値が PP にコピーされており、PP をベースに引き数を参照することができる（図5）。

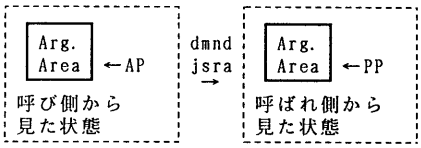


図5 引き数領域の受渡し

命令は前倒し呼びでの引き数受渡しを、高速に実現することを念頭に設計した。Cコンパイラが出すコードでは、引き数をスタックに積み、フレームポインタで引き数参照するのが簡単な方法であろう。これをそのまま前倒し呼びに拡張すると、呼ばれ側が起動する前に、引き数領域に積まれた引き数をスタックに積みなおし、リターン前に積み戻す作業が必要になる。このコピーの負荷をなくすために、専用レジスタで引き数領域を直接参照するようにした。また並列・順序呼出しの互換性を保つため、dmnd, jsraともに同じ引き数リンクの方式をとった。jsraのような手続き呼出し命令を、高級言語用に持つ CISC プロセッサもある。

(2) 引き数領域の確保

引き数領域は、発注側と受注側のデータの橋渡しをするものであるから、呼出しから受取りまでの間存在していればよい。前倒し呼びでは、呼出しと受取りは transfer 変数が宣言されたブロック内に制限される。このことから、コンパイル時に transfer 変数一つに引き数領域一つを割り当てれば良いことになる。順序呼出しの場合は、呼び側でスタックに確保すれば簡単に実現できる。

(3) 引き数領域の構成

引き数領域は引き数の他に、返り値領域、遅延代入情報、及びこれらへのポインタで構成する（図6）。遅延代入情報は受取り後に代入をするため、代入先のアドレスと代入サイズなどを含む。

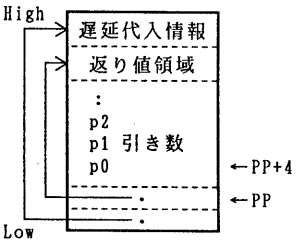


図6 引き数領域の構成

7. 物理的計算機へのマッピング

2. と3. で述べた通り、C//のプログラムに現れるのは、論理的な関数とそのインスタンス、及び論理的な変数である。実際にこれらがどのプロセッサエレメントに置かれるかはプログラム上には現れない。この間のギャップを埋める機構が必要である。

7.1 関数と関数クローン

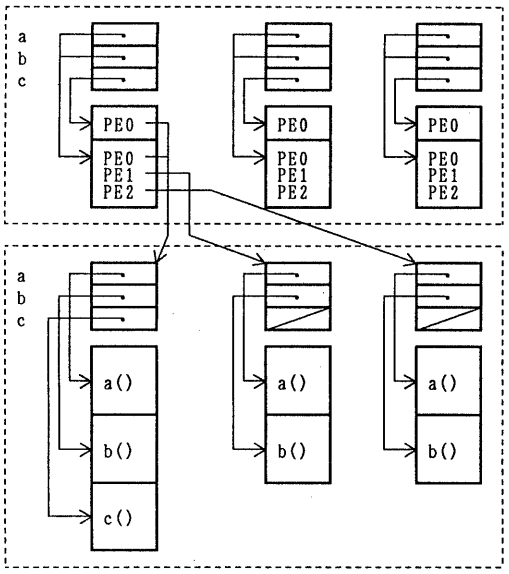
Cにおいて論理的な関数コードを代表するものは、関数名である。その値は実行系によって異なるが、一般的には関数コードの先頭アドレスであろう。しかしC//では、同じ関数に対して関数コードが複数存在するので、関数名の値を先頭アドレスとすることはできない。そこで『クローン表』を用意する必要がでてきた。これは関数ごとに関数クローンがどこに置かれているかを示す表である。どのクローンに発注するかは、実行時にこの表を参照して決定する（7.2参照）。関数名の値は、クローン表のエントリを指せばよい。

クローン表の実装方法は様々である。しかし実行時になるべく軽い負荷で参照をしようと考えると、各プロセッサエレメントのホームメモリに分散して持たせるべきであろう。図7はこのような考えに基づいた例で、クローン表を、クローンの存在するプロセッサ表と、クローンのアドレス表に分割し、すべてのプロセッサエレメントに配置したものである。関数名の値は、これらの表のオフセットにする。ラウンドロビン程度の簡単なスケジューリングならば、ホームメモリへの参照内で実現できる。

クローンのアドレス表は、制御構造のテンプレートベクタをそのまま利用している。これは、制御構造のテンプレートを、C//の関数クローンに対応させるためである。

7.2 関数インスタンスの分配

C//では、プログラム中で発注先のプロセッサを指定しない。そのため実行時にプロセッサにばらまかれた関数クローンの内、どれに発注するか決定しなけれ



上 :クローンプロセッサ表
下 :クローンアドレス表
a, b, c :関数名（クローン表のオフセット）
a(),b(),c():関数クローン

図7 関数クローンとクローン表

ばならない。これを行うのがスケジューリングライブラリである。スケジューリングライブラリは、入力された関数名の値から、クローン表にしたがって、関数クローンのテンプレートベクタアドレスを出力する。前倒し呼びのコード展開を、図8に示す。

```
move  #func,D0   …D0 = 関数名の値
jsr   sched      …D0 = ﾃﾝﾌﾟﾚｰﾄﾍﾞｸﾀｱﾄﾞﾚｽ
dmnd             …D0 = Rcp. ｱﾄﾞﾚｽ
move  D0, t      …transfer変数 = D0
```

MC68010 ｺｰﾄﾞ　　D0ﾚｼﾞｽﾀ の値の推移

図8 t //= func(); のコード展開

7.3 関数クローン・変数の分配

分散共有型並列計算機では、頻繁に使うデータやコードをホームメモリに置くことを想定している。データのうち自動変数は、インスタンス生成時にホームメモリの IDB ヒープにとられるので問題はない。一方、外部変数、静的変数、関数クローンをどのプロセッサエレメントに置くかはアプリケーションに依存するため、ユーザが明示的に指定できるべきである。そのため『ロード条件ファイル』でこれを指定する。ローダはこのファイルに従って、関数クローン、変数を配置し、クローン表を作成する。C//が、論理的な関数コードと変数を記述するのに対し、ロード条件ファイルはこれらの物理的な計算機へのマッピングを記述するもので、相補的な関係にある。試作段階のローダでは、すべてのプロセッサエレメントにすべてのクローンを分配し、共有変数は分配していない。

8. 実行時ライブラリ

ここでは、ユーザに提供すべきライブラリと、C//実行系の実現上コンパイラが必要としたライブラリについて述べる。

(1) 前倒し呼びの判定

C//では一つの関数を、並列、逐次どちらでも呼べるので、受注側ではどちらで呼ばれたのかわからない。しかしインスタンス起動にともなった初期化を記述するとき等は、どちらかを知る方法が必要である。そこで、ライブラリ関数でこの機能をプログラマに提供する。この関数は、スタックに積まれた帰り番地から、現在実行中の関数が前倒し呼びされたものか判定する（6.2参照）。

(2) 記憶管理

malloc等でメモリを取得する場合、確保する場所の選択が三つある。すなわち、①インスタンスのデータ領域（IDB）、②ホームメモリ、③すべてのプロセッサから等距離のメモリである。これらのどれを選択するかは、アプリケーションに依存するので、三つのメモリ取得ライブラリを用意してプログラマが選択できるようにする。

(3) 受信点アドレス管理

発注したインスタンスを受け取らずにブロックを出てしまうと、発注側インスタンス終了まで受注側インスタンスが残ってしまう。これを防ぐため、関数単位でのインスタンス消滅の管理を行う機能を、実行時ライブラリで用意する。このライブラリは、関数入口、前倒し呼び、関数出口で受信点アドレスをスタック管理するものである。コンパイラはこのライブラリの呼出しをコードに埋め込む。

(4) インスタンスの動的分配

C//で記述された論理的な前倒し呼びを、物理的なプロセッサへの発注にマッピングするルーチンを用意する。これをライブラリとしてプログラマが自由にスケジューリングを記述できるようにする（7.2参照）。

9．応用例

筆者らの研究室では、信学論ＤⅡに既報の『人工視覚における運動視差の機能レベル模倣』[5]の研究が進行していたので、同文献６．１の『エッジ追跡』及び６．２の『空間リストの整形』を応用プログラムのサンプルとした。このアルゴリズムを画像処理、プログラミングとも堪能であり、かつＣ//の設計には直接関与していない修士課程学生に、Ｃ//によって記述させた。アルゴリズムは既にＣ言語によって記述されているが、それを参照することなしに、アルゴリズムと教科書だけを与えて独自に記述させた。

明度こう配の計算は、単純な画面分割で行った。輪郭追跡を始めるための追跡開始点の探索は面積分割で行うが、開始点が見つかると、分割の境界に関係なく追跡を続ける。このとき際どいアクセスにより二重追跡される可能性もあるが、画素の排他はしなかった。このようなことが起こっても、異常な輪郭として空間リストの整形処理のときに削除される。空間リストの整形処理では、セグメントのリスト毎に関数インスタンスを割り当てた。このようにここで使用された並列化手法の多くは人海戦術型であった。それに対して、機能分割型はせいぜい全体の半分程度の関数の中に、２～３本の制御フローを得る可能性があったにすぎない。また、 amomg 文や /!演算子は出現しなかった。このような性質は画像という対象に依存していることは言うまでもない。

使用感については、微分フィルタのコンボリューションのプログラミングを終えたあたりでＣ//に慣れ、以後この枠組みで問題を考えられるようになり、一度慣れると再び逐次型言語に戻りにくくなるとの感想を得ている。

なおここで得られたＣ//のプログラムは、プロセッサエレメント４個の単純な共有メモリ型計算機上で実行し、動作の確認をしている。この計算機は文献[1]に準ずる制御機構をもつが、人海戦術型のプログラムは、3.1～3.9倍の高速化を得た。

１０．おわりに

並列手続き呼出しを持った言語Ｃ//と、発注／受領型並列計算機への言語処理系の実装、及び画像処理に利用した経験を報告した。現在Ｃ//の言語処理系は、クロスコンパイル、アセンブリソースリンクによって実現している。コードは、逐次計算機用のエミュレーション命令で実行を確認できた。５．で述べた実行機は一次、二次試験を終え、実用機を開発中である。また例外処理を利用したデバッグ、保護機構の検討も行っている。

謝辞

本研究をすすめるにあたって、日頃ご指導いただいている本学斎藤延男教授に深謝いたします。また実行系を共同開発した本学大学院田村仁氏、Ｃ//で応用プログラムを記述した本学大学院（現在松下電器産業）八十島広至氏に深謝いたします。

参考文献

[1]冨澤眞樹：“発注／受領型並列計算機の制御機構とハードウェア”，並列処理シンポジウムＪＳＰＰ'91論文集，pp221-228(1991).

[2]Conway, M.E.：“A Multiprocessor system design”, Proc. AFIPS Fall Jt. Computer Conf., vol.24, pp.139-146(1963).

[3]Halstead Jr., R.H.：“ Multilisp : A language for concurrent symbolic computation”, ACM Trans. Program. Lang. & Syst., vol.7, no.4, pp.501-538 (1985).

[4]田村，冨澤：“発注／受領型並列計算機の例外処理とデバッグ支援機構”，情処研報，ARC-89-18(1991).

[5]小笠原，生出，阿刀田，五十嵐，斎藤：“人工視覚における運動視差の機能レベル模倣”，信学論(Ｄ－Ⅱ)，J74-D-Ⅱ(1991-07).