

推論プロセッサ UNIREDIⅡの命令セット

島田 健太郎, 下山 健, 清水 剛, 小池 汎平, 田中 英彦

東京大学 工学部

概要

UNIREDIⅡは並列マシンの要素プロセッサとして、コミットイド・チョイス型言語 FLENG を効率良く実行するために設計された推論プロセッサである。並列推論マシン PIE64 の推論ユニット (IU) において、ネットワーク・インターフェース・プロセッサ (NIP)、汎用プロセッサ SPARC と協調動作し、FLENG の処理を行なう。UNIREDIⅡは多重コンテキスト処理の機能を持ち、3本のメモリバスを備えるなど、特色あるプロセッサ・アーキテクチャを採っている。本論文では、FLENG の並列実行を効率化することを目標として設計された命令セットについて述べる。UNIREDIⅡの命令セットでは、強力なデレファレンス命令、並列実行に対応したバインド命令などを備えながら、すべての命令が1ワードの固定長で構成され、1パイプライン・スロットで実行される。

The Instruction Set of the Inference Processor UNIREDIⅡ

Kentaro Shimada, Takeshi Shimoyama, Takeshi Shimizu,
Hanpei Koike, Hidehiko Tanaka

Department of Electrical Engineering, Faculty of Engineering,
University of Tokyo

Hongo 7-3-1, Bunkyo-ku, Tokyo 113, Japan

Abstract

UNIREDIⅡ is a logical inference processor. it executes a committed choice language FLENG efficiently, and is used for an element processor of parallel machines. In the Inference Unit of the parallel inference machine PIE64, UNIREDIⅡ cooperates with NIP, the Network Interface Processor, and the SPARC processor. UNIREDIⅡ has a dedicated processor architecture, such as multi-context processing mechanism and three memory busses. In this paper, the instruction set of UNIREDIⅡ is proposed. This instruction set is designed for efficient FLENG execution. Including useful dereference instructions and synchronous bind instructions, all instructions of it have 1-word fixed length, and are executed in 1 slot of the internal pipeline.

1 初めに

我々は現在並列推論マシン PIE64[1]の製作を進めている。PIE64では、対象言語として Committed Choice 型言語 FLENG[5]、及びその上位言語として並列オブジェクト指向言語 FLENG++[6]で書かれたプログラムを実行する。PIE64は64台の推論ユニット(Inference Unit: IU)を2系統の回線交換ネットワークで結合した構成を採る[3]。各IUにはネットワークと接続しFLENG向きの高度な通信機能を提供するNIP(Network Interface Processor)[2]、IU内でのFLENGの実行処理を行なう推論プロセッサ UNIREDD、及び全体の管理を行なう汎用プロセッサ SPARCがある。我々はこの推論プロセッサ UNIREDDについて、先に新たに効率化した第2版(UNIREDDII)のアーキテクチャの概要設計を行なった[4]。現在、詳細設計を進めているが、ここではその中で設計された命令セットを中心に、UNIREDDIIのアーキテクチャを説明する。

2 並列推論マシン PIE64

初めに PIE64 における UNIREDDII の使われ方について簡単に説明する。

並列推論マシン PIE64 は、64 台の推論ユニット(IU)を2系統の自動負荷分散機能を持つ回線交換ネットワークで接続した構成を採る[3]。各IUには、ネットワークと接続しIU間の通信/同期機能をFLENGの実行に適した高度な形で提供するNIP(Network Interface Processor)、FLENGの実行処理を行なう推論プロセッサ UNIREDDII、及び全体の管理や入出力処理を行なう汎用プロセッサ SPARCがある。PIE64における実行処理は、各々のIU内ではこれらのプロセッサによって機能的に分担され、並列に行なわれる。

IU内において、これら3種のプロセッサ間の通信は専用のコプロセッサ・コマンドバスを用いたコマンド/リブライの送受の形で行なわれる。例えば、他IUへのリモートなメモリアクセスは、UNIREDDIIやSPARCからリモート・メモリ・アクセスのコマンドがコマンドバスによってNIPに発行され、NIPはIU間ネットワークを通じて所定のメモリアクセスを実行、その結果を再びコマンドバスによってリブライとして返す。また各IUのローカル・メモリはこれらのプロセッサで共有

されており、十分なデータ転送のバンド幅を確保するため4つのバンクに分割され、各々を3系統のメモリバスでアクセスできるようになっている。

3 UNIREDDIIのアーキテクチャ

3.1 UNIREDDIIの特徴

推論プロセッサ UNIREDDIIは Committed Choice 型言語 FLENG を効率良く実行すること、及び並列マシンの要素プロセッサとして適した構成とすることの二つを目標として設計された専用プロセッサである。その特徴としては次のようなことが挙げられる。

- 記号処理向きの機能としてタグ・アーキテクチャを採用、FLENGの処理を効率化している。
- 命令バス、メモリ読み出しバス、メモリ書き込みバスの三つの分離したメモリバスを持ち、バンド幅の広い並列メモリアクセスを行なう。
- 多重コンテキスト処理によって動的なパイプライン充足率の向上を行なう。
- コプロセッサ・コマンドバスを持ち、NIP、SPARCとの間でのコマンド/リブライの通信プロトコルをサポートしており、これらのプロセッサと協調動作を行なう。
- FLENGの処理を効率化する強力な命令セットを持つ。

UNIREDDIIでは、その大きな特色として、各パイプライン・スロットに複数のコンテキストからの命令を動的に投入して実行する多重コンテキスト処理を行なっていることがある。我々は、これはFLENGのような細かい並列性を有する言語を実行するには適した形態であると考えている。従ってUNIREDDIIでは、汎用のRISCプロセッサでよく用いられるディレド・ジャンプは行なっていない。またハードウェア量の制約から、現在同時に処理できるコンテキスト数は4となっている。

3.2 データ形式

UNIREDDIIで扱うデータ形式を図1に掲げる。タグ部及び値部を合わせて1ワード32ビットで構成されている。PIE64では各々のIUに分散してメモリが置かれることに対応して、リスト等へ

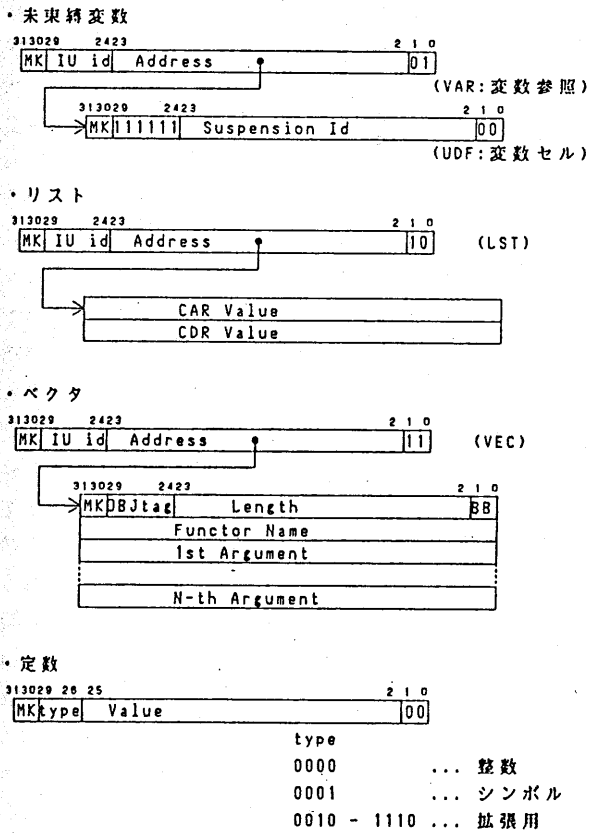


図 1: UNIREDIに於けるデータ形式

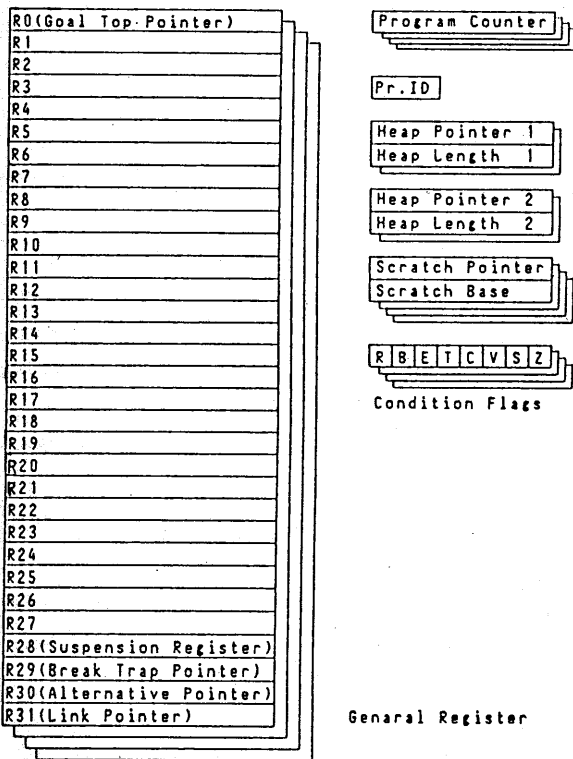


図 2: UNIREDIのレジスタ・セット

のポインタはIU番号(IU id)とIU内アドレスの組で表現されている。また未束縛変数は、Committed Choice型言語特有の処理としてサスペンドしたゴールの情報を記録するために、未束縛を表す実体(UDFセル)とそれへの参照(VARセル)とで表現されている。リストなど構造データ中に変数がある場合には、変数の実体(UDFセル)を構造データ中に埋め込むことも行なわれる。

3.3 レジスタセット

UNIREDIでは命令で意識されるレジスタセットとして、図2のようなものを持っている。多重コンテキスト処理に対応して、各コンテキスト毎にプログラム・カウンタ、32本の汎用レジスタ、リモートな構造データのローカル・コピーを作るためのスクラッチ領域管理レジスタ、フラグ・レジスタを別々に持つ。コンテキスト間で共通なレジスタとしては、ヒープ・レジスタ、プロセッサIDレジスタ等がある。また、汎用レジスタの一部にはリンク・レジスタ(R31)等、命令で特に意味付けされているものがある。

4 命令セットの特徴

UNIREDIの命令セットを表1に掲げる。これらの命令セットはFLENGの並列実行を効率化することを目的として設計された。主な特徴として、

1. 強力なデレファレンス命令
2. 最低1クロックで実行可能なバインド命令
3. タグの検査を同時に行う算術/論理演算命令
4. リモート/ローカルの区別しない統一的なメモリアクセス命令
5. マーク/スキャンアルゴリズムに対応したガベージ・コレクション用命令
6. NIP, SPARCとの協調動作を支援するコプロセッサ間通信命令

等の命令を備えていることが挙げられる。

また、すべての命令は1ワード32ビットの固定長であり、パイプラインの1スロットで実行される。

5 主な命令

次に主な命令の動作を説明する。

表 1: UNIREDIIIの命令セット

Dereference	derf dcil dfcv	dereference dereference and check list dereference and check vector and load top	dfcl dcvl dfcc	dereference and check list and load car dereference and check vector dereference and check constant
Execute	exec exll exvl	execute execute on list and load car execute on vector and load top	exel exev exct	execute on list execute on vector execute on constant
Manipulate Structure	cpir cvtp	copy if remote check vector top	cprr cvtr	copy if remote with register check vector top with register
Load / Store	ld ldst st stim	load load and store store store immediate	tgld tlds sudf	tagged load tagged load and store store undefind code
Active Unification	bind cvos	bind variable check variable order and swap	bdim	bind with immediate
Heap Allocation	allc	allocate		
Flow Control	jump jcmp jtag jrmt jcc	jump jump on compare jump on tag jump on remote pointer jump on flag condition	call jncp jntg jloc stop	call jump on not compare jump on not tag jump on local pointer stop
Coprocessor	cpcm ends fork	send coprocessor command send suspend end command send new goal command	susp fail succ	send suspend command send fail command send success command
Garbage Collection	ldsm jmrk	load and store mark jump on mark/stop contidion	stmm	store with modified mark
Set	setc seta clrf	set constant set alternative pointer clear condition flags	sett setf	set mark and tag set condition flags
Arithmetic and Logical	add sub and xor rol rcl shl asr	add subtract bit-wise and bit-wise exclusive or rotate left rotate with carry left shift left arithmetic shift right	adc sbb or not ror rcr shr	add with carry subtract with borrow bit-wise or bit-wise not rotate right rotate with carry right shift right
Condition Check	grt less abov belw equ plus ovf cty	greater than less than above below equal plus overflow correct type	geq leq abeq bleq neq mins nov ity	greater than or equal less than or equal above or equal below or equal not equal minus not overflow incorrect type
Management	spid shp exhp ltp ldf	set pid register set heap pointer exchange heap pointer set load from temporary area pointer load from flags	pidt lhp stp stf	preset pid and tag load from heap pointer set temporary area pointer store flags

5.1 デレファレンス命令

5.1.1 デレファレンス命令の基本機能

UNIREDIIIの命令セットで最も特徴的な命令としては、変数の参照の手続きを行なうデレファレンス命令がある。一般に論理型言語の処理において、変数の参照が続く間その参照先を読み出すというデレファレンス操作は、複数ステップのシーケンスを要するものであるが、UNIREDIIIIではすべての命令を基本的にパイプラインの1スロットで実行することとしたので、このデレファレンス命令の実装は次のように行なわれた。即ちUNIREDIIIIでは、デレファレンス命令をレジスタの内容が変数への参照であるかどうか調べる条件ジャ

ンプ命令と、その参照先のメモリを読み出すロード命令との複合命令であるとして実現し、レジスタの内容が変数への参照であったら、その参照先のメモリを読み出すと同時に自分自身へジャンプを行なうようにした。このようにすることによって、命令のフェッチ回数は増えるが(自分自身へジャンプした後、再び同じ命令が読み込まれる)、多重コンテキスト処理により他コンテキストの命令がデレファレンス・ループの間をかいぐって実行されることになり、一つのコンテキスト中のデレファレンス命令で他のコンテキストの処理が必要以上に遅延させられることがない。

またUNIREDIIIIのデレファレンス命令では、FL-ENGのような Committed Choice 型言語に特有

のサスペンドの処理を効率化するための機能も備えている。サスペンション・レジスタがそれで、デレファレンスしたメモリの内容がまだ未束縛の変数であったら、そのアドレスをサスペンション・レジスタ(汎用レジスタのR28)に格納するというものである。Committed Choice型言語の処理において一つのゴールと複数の候補節のヘッドとのマッチングを試みる場合、これを逐次的に行なうインプリメントでは、各々の候補節でサスペンドが生じる度に一般にスタック等を用いてその原因となった変数を記録しておかなければならないが、特に候補節の数が少ない場合に、このスタックをレジスタに展開することによって効率化しようとするものである。またもう一つの利点として、多数の候補節がある場合でもそれらが決定的に選択される時には、実際には初めの1回のサスペンドが生じるのがわかれば充分であるので、このようにサスペンション・レジスタを用いると大きな効果が期待できる。

UNIREDIではこのサスペンション・レジスタによる効率化に対応して、デレファレンス命令の内、更にデレファレンスした結果についてそれが例えばリストへのポインタであるかどうかなどの検査を行なう機能を持つものについては、デレファレンスした結果が未束縛の変数であった時(即ちサスペンドが生じた時)のジャンプ先と、デレファレンスした結果に対する別の検査が失敗した時のジャンプ先を別々に指定できるようになっている。具体的には、一方を命令中のオペランドで、他方をAlternative Pointer(汎用レジスタのR30)の内容で指定可能である。

5.1.2 デレファレンスに関する複合命令

UNIREDIでは、デレファレンス系の命令に対し、1命令1パイプライン・スロットの原則の上で効率化のためできるだけ多くのパイプライン・ステージを有効利用できるように、いくつかの複合命令を用意した。これらには、例えばデレファレンスした結果がリストへのポインタであるかどうかを調べるdereference and check list(dfcl)命令、更にリストへのポインタであった時にその先頭の1ワードを読み出すdereference and check list and load car(dcll)命令などがある。このような命令は構造データ操作を効率化する。また他にtail recursionを行なうexecute命令とこれらのデレファレンス系命令との複合命令もあり、ゴールが再帰的に処理される場合の効率化を行なっ

ている。

5.2 バインド命令

変数の束縛を行なうバインド命令は、指定された変数の内容を読み出し、まだ未束縛であったらロックを掛けて指定された値を書き込むという処理を、二つのメモリ・アクセス・バス(読み出しバス、書き込みバス)を用いてパイプラインの1スロットで実行する。この時未束縛であった変数の実体(UDFセル)に別のゴールのサスペンドが記録されていたら、NIPに対しそれらをアクティブイトするコマンドを発行する¹。また、指定された変数がリモートな変数の場合、初めからNIPに対して変数のリモート・バインドのコマンドを発行する。

なお、未束縛な変数どうしの単一化をPIE64のような分散環境で行なう時には、変数参照がループ構造になってしまわないように、束縛の方向付け等の工夫が必要であった[8]。UNIREDIでは、二つのレジスタの内容をポインタとして比較し、そのIU idも含めたアドレスの大小によって交換を行なうcheck variable order and swap(cvos)命令を用意して、そのような処理を効率化した。

5.3 算術 / 論理演算命令

UNIREDIは、整数演算の組み込み述語の処理やアドレス計算等を能率良く行なうため、整数に関する算術 / 論理演算命令を持っている(表1)。これらの命令は同時に指定されたレジスタ・オペランドのタグを検査する機能も備えており、例えば整数演算命令で指定されたレジスタのタグ部が整数型でなかったらタイプ・エラー・フラグのセットを行なうようになっている²。また、オペランドに仮定されるデータ型が整数型であるか、あるいはポインタ型であるか(アドレス計算時)等の指定も可能である。

5.4 ガベージ・コレクション命令

ガベージ・コレクション用の命令として、UNIREDIではマーク / スキャンアルゴリズムに対応した命令をいくつか備えている。これらの命令は

¹NIPがこのコマンドを受けるのは、サスペンドしている中に他のIUにあるリモートなゴールがある可能性があるからである。

²ハードウェアの簡単化のため、このような場合についてトランプ処理は行なっていない。

指定したオペランドのマーク部 (bit 31, 30 の 2 ビット) を検査し、あるいはそのビットのセット / リセットを行なう。特に多重コンテキスト処理で一時に複数のマーキング処理も行なえるように、あるメモリ中の 1 ワードのマーク部に対して排他的な test and set の処理を行なう load and store mark(ldsm) 命令もあり、これはバインド命令と同じようにパイプラインの 1 スロットで実行可能である。

5.5 コプロセッサ間通信

UNIREDIIIにおいて、NIP、SPARCと協調動作するためのコプロセッサ・コマンドは、条件によって暗黙に発行されるものと、明示的にコプロセッサ・コマンド命令 (cpcm 命令) によって発行されるものの 2 種類がある。

暗黙に発行されるもので典型的なものには、先のバインド命令のように他の IU のリモートなメモリに対してアクセスが起きた時に、これを NIP へのメモリアクセス・コマンドの発行に置き換えるものがある。UNIREDIIIのメモリ・アクセス命令では原則的としてこの機能を備えており、例えばリモートなメモリに対するロード命令では NIP に READ1 コマンド [2] を、デレファレンス命令では NIP に Deref コマンド [2] を自動的に発行する。これによって統一的なメモリ・アクセスを実現している。

一方、明示的に発行されるコプロセッサ・コマンドは、ゴール管理のために管理プロセッサ SP-ARC に対して発行されるものである。これらのコマンドにはゴールの実行終了やサスペンドを通知するものがあり、それぞれ cpcm 命令の一形態である succ、fail、susp 命令等で発行される。

6 命令セットの評価

6.1 コンパイル例

前節までで述べた命令セットを用いて、FLENG のプログラムが具体的にどのようにコンパイルされるかの簡単な例として、append の例を図 3 に掲げておく。この例では append が再帰的に実行される場合、そのループを構成する命令数は 8 であり、パイプラインの 8 スロットで実行可能である。現在、FLENG プログラムのコンパイル及び最適化処理については、更にいろいろな戦略を検討中である [9][7]。

```
append([H | T], X, Y) :- Y = [H | Z], append(T, X, Z)
append([], X, Y) :- X = Y.
```

```
append:
    seta    $suspend, ap
    dcll    r1, $1, r4                ; [H |
$2:
    tglld   [r1 + 1], r1              ; T]
    allc    s, lsl, 2, r5            ; [
    st      r4, [r5], r6              ; H |
    sudf    [r5 + 1], r6              ; Z]
    bind    r5, r3, r7                ; Y = [H | Z]
    jntg    r7, udf, $check1
    mov     r6, r3                    ; Z)
    exll    r1, $2, r4                ; [H |
$1:
    dfcc    r1, nil, $fail            ; []
    derf    r2, $ + 1                 ; X = Y.
    derf    r3, $ + 1
    cvos    r2, r3, r2, r3
    bind    r2, r3, r7
    jntg    r7, udf, $check2
    succ    gtp, mp
    stop
```

図 3: append のコンパイル例

6.2 性能予測

最後に、ソフトウェア・エミュレータにより UNIREDIIIの命令セットの評価及び性能予測を行なったので、その結果を表 2 に掲げる。

評価に用いたプログラムは、append 100 (長さ 100 のリストに対する append)、naive reverse 30 (長さ 30 のリストの naive reverse)、quick sort 50 (長さ 50 の整数値のリストの quick sort)、primes 50 (50 までの素数の計算) である。表 2 で最大性能は、クロック周波数 10MHz で多重コンテキスト処理によりパイプライン・ブレイクが起きなかったとした時の予測性能である。また、実行命令数の欄には、パイプライン・ブレイクの原因となる可能性のあるジャンプ命令について、その実行回数と全体の命令実行数に対する割合も示した。実行可能なコンテキスト数が 1 の時には、多重コンテキスト処理が行なわれないので、パイプライン・ブレイクが生じて、ジャンプ命令が 1 回実行される度に 3 クロックのペナルティがかかる。そこで、そのような場合の性能低下を見積もるために、実行コンテキスト数が 1 の時 jump 命令を考慮した性能値も求めてみた。実行コンテキスト

表 2: サンプル・プログラムによる性能予測

プログラム	コード・サイズ ¹ (word)	実行命令数		Reduction回数			最大性能 (KRPS)	jumpを考慮 した性能 (KRPS)	備考
		jump	(total)	rec. ² & call	fork				
append 100	49	818	101 (12.4%)	101	100	1	1235	901.0	
naive reverse 30	86	4426	526 (11.9%)	496	465	31	1121	826.1	
quick sort 50	136	6972	809 (11.6%)	380	279	101	545.0	404.3	
primes 50	204	19160	3985 (20.8%)	388	371	17	202.5	124.7	剰余計算もゴールに 数えた場合

¹ プログラムの命令部分の大きさ

² rec. : tail recursion により実行されたもの

```
primes(Max, Ps) :-
    gen(2, Max, Ns), sift(Ns, Ps).

gen(N, Max, Ns) :-
    greater(N, Max, Greater),
    gen1(Greater, N, Max, Ns).
gen1(false, N, Max, ![N | Ns1]) :-
    add1(N, N1), gen(N1, Max, Ns1).
gen1(true, N, Max, ![]).

sift([P|Xs], ![P | Zs1]) :-
    filter(P, Xs, Ys), sift(Ys, Zs1).
sift([], ![]).

filter(P, [X|Xs], Ys) :-
    mod(X, P, Mod), zero(Mod, Zero),
    filter1(Zero, P, [X | Xs], Ys).
filter(P, [], ![]).
filter1(false, P, [X | Xs], ![X | Ys1]) :-
    filter(P, Xs, Ys1).
filter1(true, P, [X | Xs], Ys) :-
    filter(P, Xs, Ys).

greater(N, Max, Greater) ..... NがMaxより大きかったら
Greaterにtrueを、そうでなければfalseを返す。
add1(N, N1) ..... Nに1を加算したものをN1に返す。
mod(X, P, Mod) ..... XをPで割った余りをModに返す。
zero(Mod, Zero) ..... Modが0ならZeroにtrueを、
そうでなければfalseを返す。
```

図 4: primes (素数を求めるプログラム)

数が1である時の他の性能低下の要因としては、メモリからロードされた値を直後の3命令以内に参照した場合が挙げられるが、今回はレジスタ・トランスファ・レベルのシミュレーションは行っていないので、表2には現われていない³。なお、処理されたゴールの数では、tail recursionにより同一コンテキスト内で連続して実行された個数と、負荷分散の契機となるようにあるコンテキストから他へ吐き出された(forkされた)個数も併せて調べた。

UNIREDIでは、整数の算術演算命令を備えているので、整数演算を行なう組み込み述語は基本的に命令中にインライン展開されて実行される。このため、整数演算を行なうプログラム(quick sort、primes)では、組み込み述語に相当する分だけゴールの実行数が減り、見かけ上性能値が低下している。また primes(図4)では剰余計算(組み込み述語 mod(X, P, Mod)の呼び出し)は整数除算のサブルーチン呼び出しで行なわれており、このサブルーチンの実行に120ステップ程度かかるので、剰余計算を一つのゴールとしてそれだけを実行すると、これに対する見かけの性能は10MHzクロックで最大83KRPS(Kilo Reduction Per Second)となる。

このようなことについては、他にも例えば浮動小数点演算などのある応用プログラムでは問題となるが、現在PIE64システムでは推論ユニット内に汎用プロセッサSPARCとそのFPUを搭載する予定であるので、時間のかかる数値計算等はそ

³このような場合に性能低下を抑える対策としては、コンパイラによる命令の並び替えなどの最適化を予定している。

れらに任せて、UNIREDIは推論処理に専念させることも可能である。

7 終りに

Committed Choice 型言語 FLENG の並列実行を効率化する推論プロセッサ UNIREDIの命令セットについて述べた。現在 UNIREDIは詳細設計を進めており、最終的には CMOS ゲートアレイとして実現される予定である。今後の課題としては、レジスタ・トランスファ・レベルのシミュレーション等も含めたプロセッサ・アーキテクチャの評価、PIE64 上に実装した時の NIP、SPARC との協調動作の評価、及び 64 台での並列動作の評価などが挙げられる。

なお、本研究は文部省特別推進研究 No.62065002 による。

参考文献

- [1] Koike,H.and Tanaka,H.: "Multi-Context Processing and Data Balancing Mechanism of the Parallel Inference Machine PIE64" Proc. of Fifth Generation Computer Systems, ICOT, November 1988.
- [2] 清水, 小池, 島田, 田中: "並列推論マシン PIE64 のネットワーク・インターフェース・プロセッサ" 並列処理シンポジウム JSPP '89, 情報処理学会, 1989 年 2 月
- [3] 高橋, 田中: "並列推論マシン PIE64 におけるインターコネクションネットワークの作成と評価" 情報処理学会計算機アーキテクチャ研究会 76-1, 1989 年 5 月
- [4] 島田, 下山, 清水, 小池, 田中: "推論プロセッサ UNIREDIのアーキテクチャ" 情報処理学会計算機アーキテクチャ研究会 77-2, 1989 年 7 月
- [5] Nilsson,M. and Tanaka,H: "FLENG Prolog - the Language which turns Supercomputers into Prolog Machines" Proc. of Japanese Logic Programming Conference '86, ICOT, June, 1986.
- [6] 中村, 小中, 田中: "並列論理型言語 FLENG に基づいた並列オブジェクト指向言語 FLENG++" 日本ソフトウェア科学会, オブジェクト指向計算に関するワークショップ WO-OC'89, 1989
- [7] 下山, 小池, 田中: "Committed Choice 型言語 FLENG-- 上の最適化コンパイラ" 日本ソフトウェア科学会 第 6 回大会, C1-2, 1989 年 10 月
- [8] 島田, 小池, 清水, 田中: "PIE64 上での FLENG 実行方式" 情報処理学会第 37 回全国大会 5N-6, 1988 年 9 月
- [9] 下山, 島田, 小池, 田中: "FLENG コンパイラの最適化処理" 情報処理学会第 39 回全国大会 4Q-6, 1989 年 10 月
- [10] 島田, 下山, 清水, 小池, 田中: "推論プロセッサ UNIREDIの命令セットの概要" 情報処理学会第 39 回全国大会 4W-3, 1989 年 10 月