計算機アーキテクチャ 79－5
（1989. 11. 20）

推論プロセッサUNIREDII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

UNIREDIIは並列マシンの要素プロセッサとして、コミッティド・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。UNIREDIIは多重コンテク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。UNIREDII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 UNIREDII

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

UNIREDII 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, UNIREDII cooperates with NIP, the Network Interface Processor, and the SPARC processor. UNIREDII has a dedicated processor architecture, such as multi-context processing mechanism and three memory busses. In this paper, the instruction set of UNIREDII 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 UNIRED、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 UNIRED について、先に新たに効率化した第 2 版 (UNIREDII) 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、UNIREDII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 UNIREDII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 UNIREDII、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 / リプ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、UNIREDII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プライとして返す。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、充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 UNIREDIIのアーキテクチャ

3.1 UNIREDIIの特徴

推論プロセッサ UNIREDII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。
- 多重コンテク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 / リプ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。

UNIREDII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クストからの命令を動的に投入して実行する多重コンテク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 UNIREDII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クスト数は 4 となっている。

3.2 データ形式

UNIREDII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

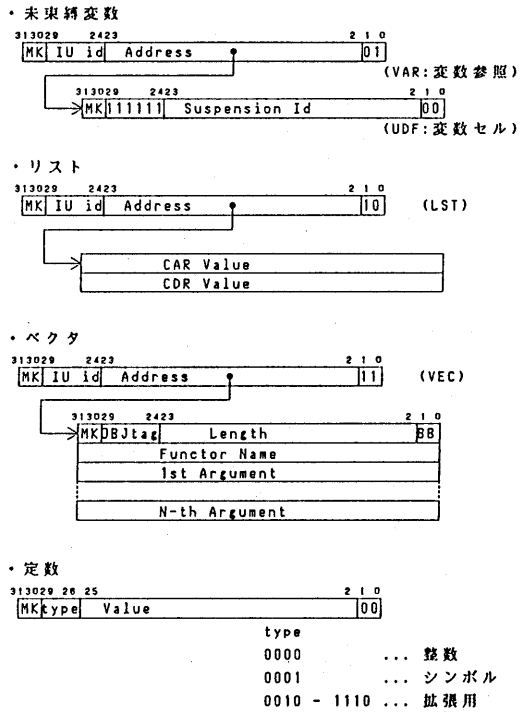

図 1: UNIREDIIに於けるデータ形式

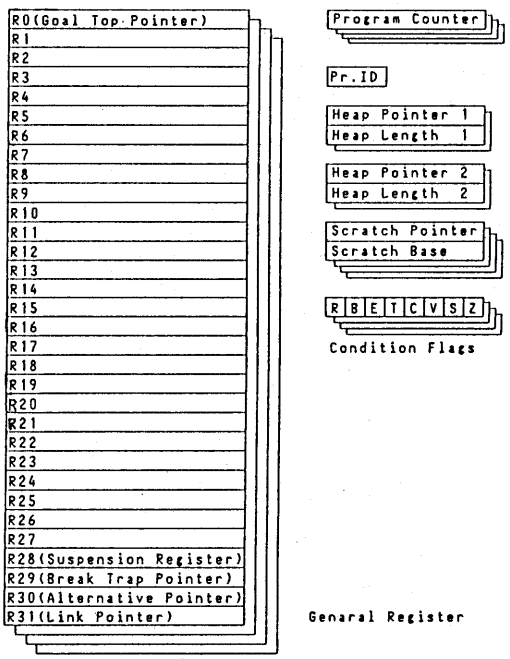

図 2: UNIREDIIのレジスタ・セット

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

UNIREDII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。多重コンテクスト処理に対応して、各コンテク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。コンテク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

UNIREDII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: UNIREDIIの命令セット

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

UNIREDIIの命令セットで最も特徴的な命令としては、変数の参照の手繰り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、UNIREDII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UNIREDII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)、多重コンテクスト処理により他コンテクストの命令がデレファレンス・ループの間をかいくぐって実行されることになり、一つのコンテクスト中のデレファレンス命令で他のコンテク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UNIREDIIのデレファレンス命令では、FLENG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。

UNIREDⅡ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

UNIREDⅡ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ベ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]。UNIREDⅡ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

UNIREDⅡ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、UNIREDⅡ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

[1] NIP 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ッ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。特に多重コンテク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

UNIREDII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。UNIREDII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SPARC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:
        tgld    [r1 + 1], r1            ; T]
        allc    s, lst, 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UNIREDII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で多重コンテクスト処理によりパイプライン・ブレー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・ブレー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。実行可能なコンテクスト数が1の時には、多重コンテクスト処理が行なわれないので、パイプライン・ブレークが生じて、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、実行コンテクスト数が1の時jump命令を考慮した性能値も求めてみた。実行コンテクスト

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	jump	Reduction回数 (total)	rec.[2] & call	fork	最大性能 (KRPS)	jumpを考慮した性能 (KRPS)	備考
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なっていないので、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 により同一コンテク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ク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。

UNIREDII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、UNIREDII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UNIREDIIの命令セットについて述べた。現在UNIREDII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" 並列処理シンポジュウム 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UNIREDII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 Laguage which turns Spuercomputers into Prolog Machines"* Proc. of Japanese Logic Programing 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 WOOC'89, 1989

[7] 下山, 小池, 田中: "Committed Choice型言語FLENG-- 上の最適化コンパイル" 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UNIREDII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