TR-675

並列推論マシンPIM/iプロセッサの設計

佐藤　正俊、武田　浩一、
大原　輝彦（沖）

August, 1991

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5
Telex ICOT J32964

Institute for New Generation Computer Technology

並列推論マシン PIM/i プロセッサの設計

佐藤正俊*、武田浩一、大原輝彦
masatosi@okilab.oki.co.jp
沖電気工業株式会社

概要

近年、知識情報処理の重要性が指摘されるに従い、システムの基盤として論理型言語指向の推論マシンが注目され始めている。推論マシンのアーキテクチャは、特殊目的の専用マシンでなく汎用的なマシンと位置付けられ、コンパイラ技術の向上やRISC指向の流れとあいまって、コンパイラによる最適化を重視した汎用的なアーキテクチャを基に実現される方向にある。本論文では、知識情報処理を実現する言語として選択した並行論理型言語 KL1 の特徴を基に、RISC アーキテクチャをベースに LIW を融合し、分岐処理の影響を少なくするアーキテクチャを提案する。このアーキテクチャの特徴は、LIW の空間並列性を導入し、パイプラインの時間並列性中の分岐処理によって影響を受ける部分を減らすことで、KL1 のような分岐頻度の高い言語の分岐処理の影響を最小限に抑える点である。このアーキテクチャの実現には、KL1 の特徴である (1)KL1b の機能が低い、(2) 分岐距離が近い、(3) 分岐頻度が高いの 3 点を利用している。また、LIW は KL1 と親和性が高く、分岐の影響である分岐命令数を実効的に減らすとともに複数操作同時実行率を上げる効果があった。

1 はじめに

近年、知識情報処理の重要性が指摘されるに従い、システムの基盤として論理型言語指向の推論マシンが注目され始めている [20, 5, 7]。推論マシンの研究は、逐次型推論マシンと並列推論マシンに大別できる。逐次型を中心とする推論マシンは、当初、ユニフィケーション等の専用ハードウェアによる高速化のアプローチを採ってきた。しかし、Prolog での WAM 方式の提案 [13] により、コンパイラによる最適化の重要性が再認識され、ハードウェア的には、データ型の判定機構とレジスタファイル程度の汎用的な構成で、比較的高い性能が得られるようになった [15]。一方、並列推論マシンにおいては、単なる逐次処理の並列化では効率の良い処理系の実現が難しく、並列推論マシン向きの論理型言語の研究が行なわれている [11, 1, 18]。並行論理型言語の提案は、処理系の実装を考慮して、効率良く高い並列性が得られる設計となっており、また、アーキテクチャから見ると、逐次型推論マシンでのコンパイラによる最適化の流れと同様に、言語による特殊性をコンパイラで吸収し、より汎用的なアーキテクチャを目指していると言える [16]。

本稿では、並行論理型言語ＫＬ１ [10, 6] を対象言語とする並列推論マシン PIM/i プロセッサの設計について述べる。PIM/i プロセッサ設計の特徴は、LIW の空間並列性を導入し、パイプラインの時間並列性中の分岐処理によって影響を受ける部分を減らすことで、KL1 のような分岐頻度の高い言語の分岐処理の影響を最小限に抑える点である。PIM/i プロセッサは、並列推論マシン PIM/i における要素プロセッサのプロセシング・ユニット (PU) にあたる。並列推論マシン PIM/i のシステム構成は、8 台の要素プロセッサをクラスタとする階層構造で、クラスタ内は共有バスを介した密結合で、クラスタ間はネットワークを介した疎結合で接続する構成である [19]。

次章以下では、まず、対象言語ＫＬ１の動作特性を考察し、これを基に PIM/i プロセッサの設計方針について述べ、次に我々が設計した PIM/i プロセッサを説明する。最後に、シミュレーションによる評価結果について報告し、考察を加える。

2 対象言語：KL1

2.1 KL1 の概要

KL1 は GHC[12] をベースにした並行論理型言語である。並行論理型言語である KL1 の特徴は、OR ノードにおける非決定性にあり、その非決定性とは、Prolog のような OR ノードの下すべてを試みる非決定性 (don't-know nondeterminism) では無く、テストに成功した 1 つのノードを選択する非決定性 (don't-care nondeterminism) である。この don't-care 非決定性により、KL1 は、Prolog の並列実行時に起こる複雑な多重環境管理を同期とコミットによる選択で解決し、単純で効率の良い処理方式の実現を可能としている。これは、アーキテクチャ的に見ると、専用ハードウェアの対象としていた複雑で重い機能の必要性がなくなることを意味する。

KL1 プログラムの実行 [4] は、Prolog と同じように、与えられたゴールを空にリデュースすることであるが、Prolog が逐次実行のためにスタックでゴールを管理するのに対して、KL1 では並列実行のためにプールで管理する点が異なる。KL1 のプログラムは、次の形をしたガード付き

*Design of the Parallel Inference Machine PIM/i Processor
Masatoshi SATO, Koichi TAKEDA, Teruhiko OOHARA,
Oki Electric Industry Co., Ltd.

ホーン節の有限集合である。

H :- G1, ... , Gm | B1, ... , Bn. (m ≥ 0, n ≥ 0)

ここで、|をコミットオペレータ、H,Gi及びBiを、順に、頭部、ガード、ボディと呼ぶ。ガードの実行は、don't-care非決定性より、共有変数である入力引数の観測のみが許され、引数が未定義の場合は、他のボディの実行でその値が定まるまで待ち合わせする。ボディの実行は、ガードの実行により選択された節に対して行なわれ、並列に実行できる。言い換えれば、KL1の実行は、共有変数として共有された実行環境を基に、ゴールとして表現された比較的小さなスレッドを並列に実行することであり、この時の同期はデータ依存関係によりガードにおいて取られる。

2.2 KL1b

KL1bは、PrologのWAMと同様に、仮想マシンを想定した抽象命令であり、KL1bの特徴は、処理量にバラツキがある点である。このバラツキは、並列実行を前提とした同期操作のためにKL1bがより細かい命令で実現されたり、コンテキスト切替えやスケジューリング、負荷分散、メモリ管理、プロセッサ間通信制御等も同じKL1bとして実現されたりするために生じる。そして、これら並列特有の処理は、頻度の高い分岐処理として現れる。

対象言語とするKL1(KL1b)の特徴をアーキテクチャへの要求項目として整理すると以下のようになる。

- KL1は大半が処理量の小さいKL1bにコンパイルされる。つまり、マイクロエミュレータ方式でKL1bを実行するより、RISC命令に展開し実行する方が、マイクロディスパッチ時間や命令間の最適化において有利と考える。
- マイクロエミュレータ方式での水平型マイクロプログラムが示す様に、KL1b内には命令レベルの並列性が内在する。また、KL1bのような抽象命令の設定により、定型処理(KL1bとランタイムライブラリ)をある程度静的に決められ、処理に内在する並列性を予め抽出し易いと考えられる。
- 特に、タグとデータの扱いには並列性がある。また、KL1では多くのタグの導入による処理の最適化を前提としているため、タグとデータとを独立に持ち、並列性を活かした操作の実現が有利と考える。
- KL1での処理は、並列実行のための同期処理や動的に変わるタグのチェックのために、分岐処理の頻度が高く、分岐による影響を押えるアーキテクチャが必要と考える。

3 設計方針

KL1の特徴により、RISCアーキテクチャをベースとし、KL1bやタグ操作に内在する並列性を有効利用するためにLIWを融合する方針とする。パイプラインの設計は、分岐処理の影響を少なくするために、LIWの並列性を用いてパイプラインステージ長を変えずにパイプライン段数を減らすこととする。つまり、LIWの空間並列性を導入することで、パイプラインの時間並列性の分岐処理によって影響を受ける部分を減らすこととした。この比較的少ないパイプライン段数は、KL1のような分岐頻度の高い言語を対象とする場合、動的に変化するデータ型に応じた分岐命令の影響を最小限に抑える効果が期待できる。ここで、LIWは基本的な考え方はVLIW[3]と同じであるが、同時実行の操作数を、容易に制御できる数に制限している点でVLIWと異なる。

3.1 LIWとKL1の親和性

LIWをKL1の処理に導入することは、以下のように親和性が高い。第一に、定型処理として命令レベル並列性やタグ操作の並列性が存在しているので、それをLIWの空間並列性に置き換えやすい。ここでは、並列実行の指定をタグ操作を含めた4つの実行ユニット（分岐操作、メモリ操作、演算操作、タグ操作）から最大3つの操作として選び指定することにする。この同時指定により、プログラマが自由に分岐操作やメモリ操作、演算操作を組み合わせて使用できるため、動的に変化するデータ型に応じた分岐処理が効率的に行われることになる。

第二に、分岐処理の命令フィールドとの組合せにおいて親和性が高い。つまり、LIWを実現するためには、独立に操作を指定する命令フィールドが必要であるが、KL1bでの動的に変化するデータ型に応じた分岐は近傍への分岐であると考えられるので、分岐先アドレスの持ち方を工夫し40ビットの命令フィールドにLIWの操作指定フィールドを取り込むことができる。

第三に、LIWアーキテクチャは、ユーザ可視な実行ユニットを複数提供し、複数の操作を同時に実行するので、コンパイラによる最適化の可能性をさらに広げる。これは、RISCでKL1bを展開し実行する場合、親和性が高い。

また、専用マシン特有の特殊命令のサポートがより汎用的な枠組で実現できる。専用マシン特有の特殊処理を高速に実現する場合、処理内の並列性を利用し特殊命令に因って実現されるが、RISCやパイプラインの設計にまで影響を与える結果となり、プロセッサの設計上コストが大きい。しかしLIWの場合、基本操作の組合せとして処理内の並列性を抽出できる。

これらの他に、次節で述べるように、LIWの空間並列性により時間並列性が置き換え可能であり、また、LIWでは命令内のフィールド位置で使用する実行ユニットが一意に決まり、デコード時間が短縮でき、パイプラインの設計に有利である等の特徴を持つ。

3.2 5段から3段に

PIM/iの3段パイプラインと一般的な5段[8]とを比較し、どのようにパイプライン段数を減らしたかを示す。

PIM/iの3段パイプラインは、命令フェッチ(F)、実行(E)、レジスタ／メモリ書き込み(W)からなり、実行は、命令デコード、レジスタ読み出し、演算、アドレス計算、メモリアクセス、分岐先計算を含む。5段パイプラインは、命令フェッチ(IF)、デコードとレジスタフェッチ(ID)、実行とアドレス計算(EX)、メモリアクセス(MEM)、ライトバック(WB)からなる。PIM/iでは、5段パイプラインでのEXとMEMをLIWにより別命令で実現する点とIDとEXを1ステージで実現する2点で、パイプライン段数を減らしている。

3.2.1 LIWによるEXとMEMの別命令実行

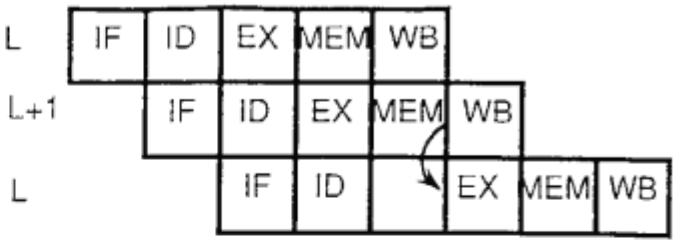
メモリアクセスはアドレス計算とメモリとのデータ転送の2つの操作から成っている。5段パイプラインでは、EXステージでアドレス計算を、MEMステージでメモリとのデータ転送を行うように構成されている。このような構成にすると、メモリアクセスの結果はパイプラインの後のほうで得られることになるため、ロード遅延の原因となる。

EXとMEMを別命令で実現する利点は、デレファレンスのようなメモリを読んでそのデータ型で分岐を繰り返すような単純な処理に現れる。図1に、5段パイプラインの場合(a)とEXとMEMを別命令で実現する場合(b)の例を示す。図において、(a)は read(a00,a00) でアドレス計算と読み込みを行なっているが、(b)では mar=r00 でアドレス計算、read(r00) で読み込みを行なっている。5段パイプラインでは、MEMステージの結果を次の命令のEXステージに渡すにはデータハザードが生じ1サイクル無駄になる。この無駄を無くすには、MEMステージとEXステージを同時実行するようにパイプラインを設計する方法がある。PIM/iのLIWの枠組では、アドレス計算と分岐操作を同時実行しながら、次命令でメモリとのデータ転送をすることで、この無駄を無くす。

3.2.2 IDとEXの1ステージ化

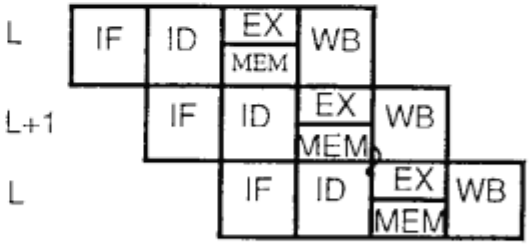
IDとEXの1ステージ化の利点は、条件分岐の実現に現れる。PIM/iでは、条件分岐を1命令で条件コード無しで実現することで、条件分岐時の命令数と遅延分岐スロット数を減らし、分岐による影響を少なくする。また、条件コードが無くなることは、コンパイラやLSIの設計での複雑さが回避できる点で有利である。

一般に、条件分岐処理に関わる命令数は、条件分岐処理を1命令で実現するか、分岐の成否を決めるTESTやCOMPARE等の命令と実際に分岐するBRANCH命令との2命令で実現するかの2つの選択がある。1命令では、複雑な条件コードの実装から逃れられる点で多くの利点があるが、[2]によると、パイプラインを構成する上で以下の不利益がある。条件分岐を1命令で構成する場合の処理は、IDステージでの命令デコード／比較するオペランドの読み込みとEXステージでの比較／分岐先の決定であり、5段のパイプラインでは、パイプラインの空きは分岐




```
L: if tag(r00)==REF  goto_canceled L .
   read(r00,r00).
```

(a) EX/MEMが別ステージ




```
L: mar=r00, if tag(r00)==REF  goto_canceled L.
   read(r00).
```

(b) EX/MEMが別命令

図 1: EXとMEMの別命令実行

命令から分岐先の決定までの2サイクルとなる。一方、2命令での構成は、1命令目のIDステージで比較するオペランドの読み込みを行ないEXステージで比較を行ない、2命令目のIDステージで分岐先の決定を行なうために、パイプラインの空きは1サイクルとなる。つまり、1命令での実現では、パイプラインの空きが1サイクル分不利である。我々は、命令デコード／比較するオペランドの読み込み／比較／分岐先の決定を、LIWにより並列に1ステージで実行するようにパイプラインを設計することで、パイプラインの空きを1サイクルのみで1命令分岐を実現する。

IDとEXの1ステージ化はパイプラインステージ長を長くすると考えられ、デコードと実行を1ステージでまとめることの良否は、命令フェッチステージ長との相対関係で決まる。しかし、我々は、パイプラインステージ長を短くする時のクリティカルパスは、命令フェッチステージ長にあると考える。つまり、命令フェッチはプロセッサチップ外をアクセスするためチップ内の処理に比べて時間を要する。また、チップ内キャッシュを導入する場合でも、キャッシュのメモリ容量（レジスタファイルの100倍から1000倍）から来るアクセスタイムの増加やキャッシュのヒット判定の必要性により、命令フェッチステージ長はパイプライン設計上クリティカルパスとなる。これに対して、LIWでは命令内のフィールド位置で使用する実行ユ

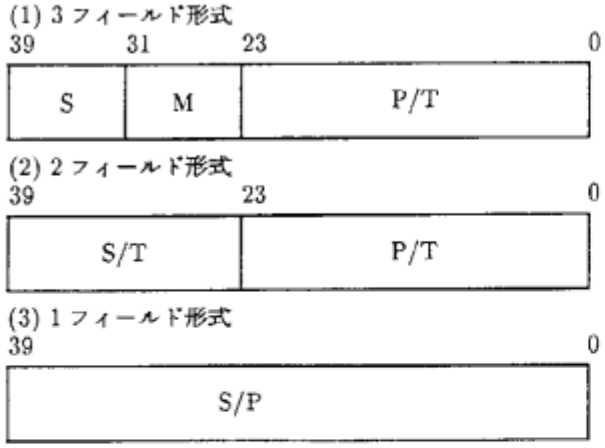


図 2: フィールド形式

ニットが一意に決まり、デコードは短く実現できる。また、実行においてはキャッシュのアクセスタイムに比べれば、短縮のための設計技術を発揮しやすい。PIM/i での各ステージ見積りでは、LIW による命令フィールドの整理と演算時間短縮のための設計（固定フォーマット命令や不必要な複雑なアドレシングモードの不採用、条件コードの不採用等）により、デコード／実行時間が短縮でき、デコード／実行は命令フェッチと同じステージ長で実現可能となっている[1]。

4 PIM/i プロセッサ

以上の設計方針より設計した PIM/i プロセッサを示す。

4.1 命令形式

分岐は、(1)6 ビットの符号付き相対分岐先を持つ 8 ビット長分岐、(2)5 ビットの間接分岐用レジスタ番号または 8 ビットの分岐時の条件を含む 16 ビット長分岐、(3)30 ビットの分岐先アドレスを保持する 40 ビット長分岐の 3 種類の分岐操作に分けて実現する。これにより、分岐処理の選び方によって、40 ビット固定長の 1 命令に同時操作を指定することが可能となる。同時指定する形式は図 2 に示す 3 つのフィールド形式で分類される。ここで、S は分岐操作、M はメモリ操作、P は演算操作、T はタグ操作を指定できるフィールドを表し、P/T は演算操作またはタグ操作のいずれかを指定できるフィールドを示している。また、以下では命令は 40 ビット固定長の単位を、操作は命令中の S、M、P、T に指定された処理を指すことにする。

[1] この設計は、パイプラインステージ長を短くしていっても適用可能であり、実際 R4000[9] では、20 ナノ程度のステージ長で命令フェッチとデコード／実行を同じステージ長で実現している。

4.2 命令セット

命令セットを各操作毎に概観する。命令セットの一覧は付録に示す。

1. 分岐操作

   分岐操作は、上記に示した命令長の違いによる 3 分類に加えて、遅延スロットの利用方法の違いによる分類がある。遅延スロットを利用する分岐操作は、通常分岐（分岐無の時は遅延スロットを実行し、分岐有の時は遅延スロットを無視する）、遅延分岐（分岐の有無に関わらず遅延スロットを実行する）、中和分岐（分岐無の時は遅延スロットを無視し、分岐有の時は遅延スロットを実行する）の 3 種類である。

2. メモリ操作

   汎用レジスタとメモリ間の転送は、メモリ操作に限る。アドレスの指定は、別操作のアドレス生成操作を使用して指定する。メモリ操作は、read/write とロック付き read/write がある。

3. 演算操作

   演算操作は、レジスタ間演算に限り、オペランドは 3 つ指定できる。操作は、算術演算、論理演算、ビットフィールド操作、即値のロード、バイト整合、バイト比較、バイト検索等がある。

4. タグ操作

   タグ操作は、タグマスク、タグ取り出し、タグ値のセット等の基本操作を用意する。

4.3 プロセッサの構成

PIM/i プロセッサのアーキテクチャを図 3 に示す。この図は、水平方向はフィールドに対応する 3 つのユニット (分岐操作、メモリ操作、演算操作 (タグ操作を含む)) を示し、垂直方向はパイプラインの 3 つのステージ (F,E,W) を示している。

1. ユニット

   分岐操作 (S) ユニットは、命令ポインタ (IP)、分岐先アドレス計算回路 (BALU) などを持ち、命令フェッチ制御を行う。メモリ操作 (M) ユニットでは、メモリ (Data Mem) とレジスタファイル (RF) 間の転送制御を行う。演算操作 (P) ユニットは、レジスタファイル (RF)、演算回路 (ALU)、値操作回路 (VOP)、タグ操作回路 (TOP)、タグ判定回路 (TM) を持ち、命令語長の関係で制限付きながらタグと値の並列操作が可能である。RF は、メモリアクセスと演算を独立に行うために、演算操作用のポート (読み出し 2 ポート、書き込み 1 ポート) の他に、ロード操作用の書き込みポートとストア操作用の読み出しポートを持っている。

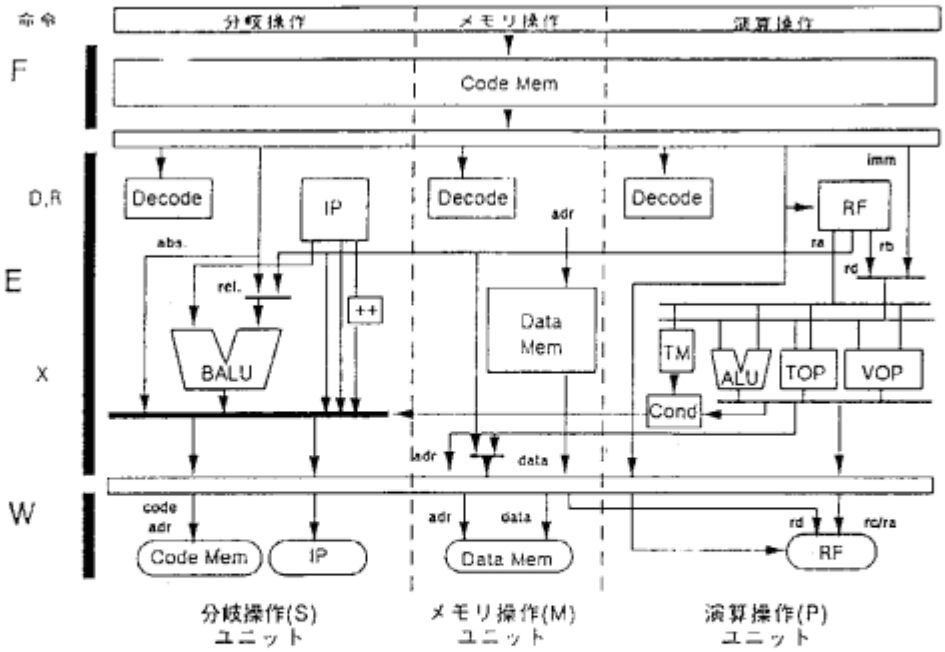


図 3: PIM/i プロセッサのアーキテクチャ

2. ステージ

Fステージは、Sユニットからの命令アドレス出力により開始され、命令バスから命令をフェッチする。ここで、命令バスはデータバスとは分離されているため、命令フェッチがデータアクセスに影響を与えることはない。Eステージは、命令デコード(D)、オペランド読み出し(R)、実行(X)を行う。このステージでは、各ユニットが並列に処理を行う。Sユニットでは、次にフェッチする命令アドレスの計算を行う。Mユニットでは、メモリまたはRFから読み出しを行う。Pユニットでは、タグ操作・判定または値の演算のいずれかあるいは両方を行う。Wステージは、IPの更新、メモリ書き込みまたはロードデータのRFへの書き込み、演算結果のRFへの書き込みをそれぞれのユニットで行う。

5 評価

5.1 評価方法

評価のポイントは、KL1の特徴によるアーキテクチャ上の選択が正しいことの確認とPIM/iで採用したアーキテクチャの効果の確認であり、KL1プログラムをPIM/iのアーキテクチャでシミュレーションすることで、これらを確認する。

KL1プログラムの実行は、KL1プログラムからPIM/iの機械語まで変換されたコード列を実行することで行う。PIM/iの機械語への変換は、KL1コンパイラでKL1プログラムを抽象マシン命令であるKL1bにコンパイルし、さらに、ポストコンパイラで各KL1bに対応するPIM/iの機械語へ変換する[17]。ここで、各KL1bの処理単位の小さなものは直接展開し、処理単位の大きなものはランタイム・ライブラリとしてローカルメモリに置き、KL1b毎に定義されたPIM/i機械語はこのランタイム・ライブラリを呼び出すことでコード・サイズを小さく抑えている。使用したランタイム・ライブラリのコードサイズは約4KWである。

表 1: ベンチマークプログラムの性格

	qk8	qu8	bup	han	pri	av.
命令サイズ	1.7	2.5	10.5	0.7	0.9	3.3
実行サイクル	3.2	5.8	4.3	2.7	3.8	4.0

命令サイズ:KiloWords, 実行サイクル:MegaCycles

シミュレーションは、キャッシュの動作や個々の命令のパイプライン処理過程をレジスタトランスファ・レベルで忠実にシミュレートするシミュレータ[14]を用いて行なう。ここで、シミュレータの構成は、プロセッサを1台とし、コード及びデータ・キャッシュは共に32KWとした。評価には、5つのベンチマークプログラム(qk8,qu8,bup,han,pr
を使用した。ベンチマークプログラムの特徴として、そのコードサイズ、実行サイクルを表1に示す。

5.2 設計方針の選択の確認

アーキテクチャ選択の確認として、(1)KL1bのレベルが低くRISC向き、(2)分岐距離が近く命令フィールドがLIWの操作に割り当て可能、(3)高い分岐頻度の影響を押える必要性の3点について評価する。第一のRISCの選択の確認のために、表2にベンチマーク実行時の実行KL1b数と1 KL1b当たりの実行命令数(PIM/iの機械語に展開された命令の実行数)及び展開された命令からのサブルー

表 2: 1 KL1b 当たりの実行命令語数

	実行 KL1b 数	KL1b 当たりの実行命令数	KL1b 当たりの呼び出し回数
qk8	12.7M	2.2	0.13
qu8	26.0M	3.0	0.13
bup	0.9M	2.3	0.11
han	2.5M	2.6	0.10
pri	2.7M	2.4	0.06
平均	9.0M	2.5	0.11

表 3: 分岐処理の出現率 (%)

	qk8	qu8	bup	han	pri	av.
条件分岐	14.0	13.3	14.6	19.5	44.7	21.2
無条件分岐	5.9	4.9	6.2	8.1	3.7	5.8
タグ分岐	15.9	9.7	15.4	4.2	9.8	11.0
分岐操作	35.9	28.0	36.2	31.8	58.1	38.0

チン呼び出し回数を示す。これによると、1 KL1b 当たりの平均実行命令数は 2.5 であり、KL1 プログラムの大半は処理量の小さな KL1b が実行されるといえる。また、KL1b からのサブルーチン呼び出し回数を見ると 0.11 回と少なく、展開命令とランタイムライブラリの実行比率がほぼ 1 : 1 であることからすると、サブルーチン呼び出しによる処理はまとまった処理であると言え、サブルーチン呼び出しによるオーバヘッドは小さいと言える。

第二は、分岐距離が近く、分岐時の分岐先アドレスの持ち方を工夫することで LIW の命令フィールドを割り当てられる点の確認のために、分岐距離の分布を図 4に示す。ここで、横軸の N は分岐距離を示し、$\log_2 N$ は命令中で分岐に必要なビット数を示している。縦軸はベンチマーク毎の全実行命令数に対する出現率を示している。ここでの分布距離は、シミュレータのデータを基に連続した分岐の分布距離を補正している。これによると、分岐のほぼ 9 割以上が 6 ビットの符号付き相対分岐命令での分岐可能範囲内であり、命令フィールドの残りのフィールドを他の操作に割り当てることは有効であると言える。

最後に、分岐の影響を知るために、各 KL1 プログラム実行時の分岐処理の出現率を表 3に示す。表が示すように、KL1 プログラムは、分岐操作の割合が非常に大きく、平均で 38% であった。その内訳では、タグ分岐を含めて条件分岐が 32% でかなり多いことが確認できる。この点より、分岐の影響を押えることは重要であり、特に、条件分岐の扱いが重要であると言える。

5.3 アーキテクチャの効果

ここでは、アーキテクチャの効果として、(1) パイプラインの 3 段化と (2)LIW による並列性の抽出について評価する。第一に、パイプラインの 3 段化は、ID と EX の 1

表 4: 遅延分岐スロットの使用率と予想使用率

	qk8	qu8	bup	han	pri	av.
1 遅延	40.4	42.4	51.1	52.5	48.6	46.8
2 遅延	12.5	10.8	9.7	2.6	5.5	8.2

ステージ化と EX と MEM の別命令化により行なわれ、分岐の影響を減らしている。分岐の影響は、分岐に関わる命令数と遅延分岐スロット数で現れる。PIM/i では、条件分岐を 1 分岐命令／1 遅延分岐スロットで設計したが、この分岐命令は LIW で他操作と同時に実行できるので実効的には分岐命令数が減ることが期待できる。全条件分岐命令数における分岐のみの分岐命令数は、5 つのベンチマークの平均で 0.20 であり、条件分岐を 0.20 分岐命令で実現していると言える。また、遅延分岐スロット数の影響については、ID と EX の 1 ステージ化に伴う設計での 1 分岐命令／2 遅延分岐スロットと比較し、遅延分岐スロット数から見た分岐の影響を考察する。表 4に、現状での 1 遅延分岐スロットの場合の遅延分岐スロット使用率と 2 遅延分岐スロットと仮定した場合の遅延分岐スロット使用率の予想を示す。表が示すように、分岐の多い KL1 のプログラムでは、1 遅延分岐スロットを利用するのが限度で、2 遅延分岐スロットは利用しきれないと言える。つまり、2 遅延分岐スロットでの実現では 8% 程度しか遅延分岐スロットを利用できないので、PIM/i の条件分岐を 1 遅延分岐スロットで実現することは、2 遅延分岐スロットに比べて分岐 (38.0%) の 91.8% (実行時間に対して 34.9%) 有利と言える。ID と EX の 1 ステージ化しない選択で条件分岐を 2 命令で実現する場合は、分岐命令の増加分が不利になる。一方、EX と MEM の別命令化は、データハザードの解決のための命令間スケジュールを行なうことと考えれば、LIW で吸収され、命令の増加による損失は無いと考えられる。

次に、LIW による並列性の抽出を考察するために、各ベンチマークのフィールド形式パターンの出現率を表 5に示す。表は、各フィールド形式の出現率と各操作フィールドの組合せの出現率を示している。S、M、P、T は、図 2で示したフィールドを表し、_ は利用されないフィールドを示す。この表より、多い組合せが [S P]、[_ M P]、[T P] であり、命令当たりの操作の同時使用率が上がっていることを示している。しかし、S が単独で使用されることも多く、さらに、最適化の余地があることも示している。各フィールド形式での同時使用率は、3 フィールド形式で 1.23、2 フィールド形式で 1.84 であり、全体で 1.44 であった。3 フィールド形式の同時使用率が 2 フィールド形式に比べ低い理由は、3 フィールド形式で指定できる分岐操作が無条件分岐に限られているためである。複数操作を同時に実行した効果は、各操作を個別に実行した場合の実行時間に比べ、約 26% の実行時間の削減である。

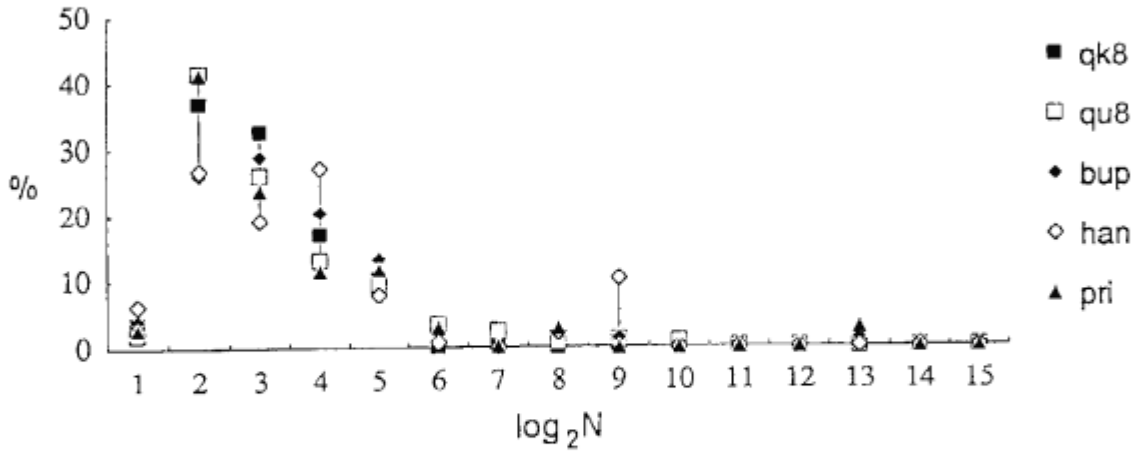


図 4: 分岐距離の分布

表 5: フィールドの出現率

	qk8	qu8	bup	han	pri	av.
3フィールド	56.1	60.0	55.8	60.6	39.8	54.5
[_ _ _]	0.0	0.0	0.0	0.0	2.4	0.0
[_ _ P]	28.9	36.1	28.5	35.3	24.6	30.7
[_ _ T]	0.2	0.1	1.2	2.6	0.6	0.9
[_ M _]	10.1	8.6	8.8	5.6	4.2	7.5
[_ M P]	14.1	13.1	11.9	9.0	4.4	10.5
[_ M T]	0.4	0.3	0.9	3.8	0.0	1.1
[S _ _]	0.9	0.4	2.9	1.4	3.0	1.7
[S _ P]	0.8	1.2	0.7	1.5	0.0	0.8
[S _ T]	0.0	0.0	0.0	0.0	0.0	0.0
[S M _]	0.7	0.2	1.0	1.4	0.6	0.8
[S M P]	0.0	0.0	0.0	0.0	0.0	0.0
[S M T]	0.0	0.0	0.0	0.0	0.0	0.0
2フィールド	37.8	35.1	38.0	31.3	56.4	39.7
[S _]	9.1	3.6	7.8	5.0	6.1	6.3
[S P]	20.3	18.2	19.7	13.9	46.0	23.6
[S T]	0.4	0.3	0.7	3.6	0.0	1.0
[T P]	8.0	13.0	9.8	8.8	4.3	8.8
1フィールド	5.9	4.9	6.2	8.1	3.7	5.9
[S]	5.1	4.5	5.1	5.4	3.7	4.9
[P]	0.8	0.4	1.1	2.7	0.0	1.0

6 おわりに

知識情報処理を実現するプロセッサアーキテクチャとして、RISC アーキテクチャをベースとし LIW を融合し、分岐処理の影響を少なくするアーキテクチャを示した。このアーキテクチャは、分岐処理の影響を少なくするために、LIW の並列性を用いてパイプラインステージ長を変えずにパイプライン段数を減らすと同時に、LIW の並列性抽出ができる点に特徴を持つ。我々は、パイプライン段数を減らすために、対象言語である KL1 の特徴である (1)KL1b の機能が低い、(2) 分岐距離が近い、(3) 分岐頻度が高いの 3 点より LIW を導入し、命令セットとパイプラインステージを整理した。これらの特徴が PIM/i のアーキテクチャにおいても現れることを、シミュレーションより確認することができた。

分岐処理の影響を少なくすることは、分岐命令数と遅延分岐スロット数を減らすことであり、PIM/i では条件分岐を、ベンチマークにおいては、0.20 実効分岐命令／0.53 実効分岐遅延で実現している。LIW による複数操作同時実行率は 1.44 であり、各操作を個別に実行した場合の実行時間に比べ、約 26% の実行時間の削減ができた。

我々は、このアーキテクチャに基づき、現在、PIM/i プロセッサを 1.2μm CMOS スタンダードセル（2 層メタル配線、配線は全自動）で、目標サイクルタイムを 100 nsec で試作中である。設計によると PIM/i プロセッサは、トランジスタ数が約１７０K、総端子数が 208 （信号線 176）である。

今後は、PIM/i プロセッサの評価を基に、KL1 のような分岐の多い言語にスーパパイプラインやスーパスカラ等の技術を導入する場合の課題を検討していく予定である。

謝辞

本研究の機会を与えて頂いた(財)新世代コンピュータ技術開発機構(ICOT)の淵一博研究所長、瀧和男第一研究室長に感謝いたします。日頃、御助言を頂く、ICOT及び沖電気のPIM研究開発メンバー諸氏に、また、多くの有益なご助言を頂いた査読者の方々に感謝いたします。

なお、本研究は第五世代コンピュータプロジェクトの一環として行なわれたものである。

参考文献

[1] K.L. Clark and S. Gregory. Parlog: Parallel Programming in Logic. *ACM Tran. on Programming Languages*, 8(1), 1986.

[2] J. A. DeRosa and H. M. Levy. An Evaluation of Branch Architectures. In *14th International Symposium on Computer Arch.*, Jun. 1987.

[3] J. R. Ellis. *Bulldog: A Compiler for VLIW Architectures*. The MIT Press, 1986.

[4] M. Sato et al. KL1 Execution Model for PIM Cluster with Shared Memory. In *The Fourth International Conference on Logic Programming*, pages 338 – 355, 1987.

[5] S. Uchida et al. Research and Development of the Parallel Inference System in the Intermediate Stage of the FGCS Project. In *FGCS 1988*, pages 16 – 36. ICOT, Nov. 1988.

[6] T. Chikayama et al. Overview of the Parallel Inference Machine Operating System (PIMOS). In *FGCS 1988*, pages 230 – 251. ICOT, Nov. 1988.

[7] A. Goto and M. Sato et al. Overview of the Parallel Inference Machine Architecture (PIM). In *FGCS 1988*, pages 208 – 229. ICOT, Nov. 1988.

[8] J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach*, chapter 6 Pipelining. Morgan Kaufmann Publishers Inc. San mateo, CA, 1990.

[9] A. Khan. 64ビット・アーキテクチャのR4000を開発. 日経エレクトロニクス, (538):171 – 185, 10 1991.

[10] Y. Kimura and T. Chikayama. An Abstract KL1 Machine and its Instruction Set. In *Int. Symp. on Logic Programming*, pages 468 – 477, Aug 1987.

[11] E. Shapiro. Concurrent Prolog: a Progress report. *IEEE Computer*, pages 44 – 58, Aug 1986.

[12] K. Ueda. Guarded Horn Clauses. Technical Report TR - 103, ICOT, 1985.

[13] D.H.D. Warren. An Abstract Prolog Instruction Set. Technical Report TR - 309, SRI International, 1983.

[14] 吉田裕一他. 並列推論マシンpim/iの開発支援環境 - シミュレータ -. In 第*41*回情報処理学会全国大会, 1990.

[15] 金田悠紀夫、松田秀雄. 逐次型推論マシンのアーキテクチャ. 情報処理, 32(4):450 – 457, 1991.

[16] 後藤厚宏. 並列型推論マシンのアーキテクチャ. 情報処理, 32(4):458 – 467, 1991.

[17] 佐藤正俊他. 並列推論マシンpim/iの開発支援環境 - 翻訳系 -. In 第*41*回情報処理学会全国大会, 1990.

[18] 市吉伸行. 論理型言語の並列処理方式. 情報処理, 32(4):435 – 449, 1991.

[19] 大原輝彦他. 並列推論マシンpim/iの概要. In 第*40*回情報処理学会全国大会, 1990.

[20] 田中英彦. 論理型言語指向の推論マシンの位置付けと開発の現状. 情報処理, 32(4):415 – 420, 1991.

分岐操作

S(39:32)	return_Delay3	-
	goto_Delay3	jr6 (jr6: 6 bit branch offset)
S(39:24)	goto_Delay2	rd
	jmp_Delay2	rd
	gosub_Delay2	rd
	jmpsub_Delay2	rd
	return_Delay3_Cond	-
	goto_Delay3_Cond	jr6
	merge_tag	rx, imm8 (imm8: 8 bit immediate)
S(39:0)	goto_Delay2	jr30 (jr30: 30 bit branch offset)
	jmp_Delay2	ja30 (ja30: 30 bit absolute jump address)
	gosub_Delay2	jr30
	jmpsub_Delay2	ja30

NOTE
Delay3: normal(通常分岐), delayed(遅延分岐), canceled(中和分岐)
Delay2: normal, delayed
Cond: tag_eq, not_tag_eq, fwd_eq(fwd:tagged full word), not_fwd_eq,
eq, not_eq, not_ovf, ovf, high_same, low, high, low_same, gt, le, test_tag

メモリ操作

M(31:24)	MemOp	rd
	LockOp	rd
	StrResOp	-

NOTE
MemOp: read, write
LockOp: read_lock, write_unlock
StrResOp: write, write_unlock, unlock

演算操作

P(23:0)	ValOp	(k,l)!ra, (n,m)!rb *or* (k,l)!ra, imm5 (imm5: 5 bit immediate)
	TagOp	tag(k,l)!ra, (n,m)!rb *or* tag(k,l)!ra, imm5 ((a,b): bit field from a to b)
	PriOp	ra, rb
	AOp	ra, rb, rc *or* ra, imm5, rc *or* ra, rb ≪ ix, rc *or* ra, imm5 ≪ ix, rc
	LOp	ra, rb, rc
	AdrOp	ra, rb ≪ ix , (mar,rc) | mar
		or ra, imm5 ≪ ix, (mar,rc) | mar
	AdrDbyteImmOp	ra, imm16 (imm16: 16 bit immediate)
	DbyteImmOp	ra, imm16

NOTE
TagOp: deposit_tag, merge_tag, extract_tag, merge_tag_imm
ValOp: deposit, merge, extract, merge_imm, merge_dynamic_imm, deposit_dynamic,
merge_dynamic, extract_dynamic
PriOp: find_first_zero, find_first_one, reverse_find_first_zero, reverse_find_first_one
AOp: *arithmetic operation*
LOp: *logical operation*
AdrOp: set_adr, set_adr_post_modify, set_adr_pre_modify
AdrDbyteImmOp: set_adr_imm
DbyteImmOp: add_signed_imm,load_low_imm,merge_high_imm