5B-7

逐次型推論マシンＣＨＩ小型化版のアーキテクチャ

小長谷　明彦　　中崎　良成　　幅田　伸一　　梅村　護
日本電気（株）Ｃ＆Ｃシステム研究所

１．　はじめに

著者らは第五世代計算機プロジェクトの一環として、論理型言語の高速実行を目指したバックエンド型推論マシンＣＨＩ[1]を小型化、改良した小型化版ＣＨＩを開発中である。本稿では小型化版ＣＨＩアーキテクチャの命令セットに焦点をあて、その特徴点である基本命令セット、組込述語テーブル、クローズコンパイル法について述べる。

２．　設計方針

推論マシンＣＨＩの開発において、著者らはWarrenが提案した命令セット[2]を基本とした機械語を設定することによりベンチーマーク性能として280K LIPSという高性能を得た。また、ＩＣＯＴの核言語第０版（ＫＬ０）に相当する機能の組込述語を直接ファームウエアで用意することにより述語を基本単位とした透明度の高いプログラミング環境を実現した。ＣＨＩのアーキテクチャ上の特徴は上記に示した論理型言語向き命令セットを備えていることの他に、動的な型判定を高速化するためのタグ付きメモリ、プログラムやアトム等のプロセス間共有データを格納するための４つのセグメント、論理型言語の実行に必要なスタック等を実現するための４つのプロセス固有セグメントを備えていることにある[1]。

小型化版ＣＨＩでは基本アーキテクチャはＣＨＩのものを踏襲し、命令セットに関して以下の観点から見直しを行った。

①システム記述で多用する単純なデータのコピーやタグの変更等の操作を高速化する。

②組込述語の使用頻度には大きな片寄りがあるため、ソフトウエアによる組込述語の実現を考慮し、ファームウエア化の効率化を図る。

③assert/retractによる動的に追加除去が可能な述語のための機械語命令をサポートし、インタプリタの高速化を図る。

小型化版ＣＨＩでは①②を実現するためにレジスタを操作対象としたタグアーキテクチャ指向の‘基本命令セット’を導入し、組込述語およびＯＳとインタプリタの一部を基本命令で実現するようにした。ただし、コンパイラは対話型環境での応答速度を保証するため、通常は述語呼出しを基本単位とした論理型言語向き命令の命令列に展開し、最適化モードで基本命令まで展開するようにした。このとき、ファームウエアで実現している組込述語と基本命令で実現している組込述語の違いをソフトウエアから不可視にするため‘組込述語テーブル’を導入した。さらに、③のインタプリタの高速実行を図るために、クローズ単位にコンパイルする‘クローズコンパイル法’を導入し、論理型命令セットの拡張を行った。

３．　基本命令セット

組込述語をソフトウエアで効率的に実現するという観点からタグアーキテクチャとしての特長を生かした以下のデータ転送命令、データ操作命令、実行制御命令を基本命令セットとして設定した。

（１）データ転送命令

データ転送命令は述語の引数がレジスタに置かれることから全てレジスターメモリ（レジスタ）転送命令とし、さらに、スタック中のデータや構造型データを容易に扱えるようにレジスタ直接、レジスタ間接、オフセット指定といった様々なアドレスモードを導入した。また、データ転送命令として、通常の１語のロード、ストア命令の他に、連続領域のメモリ間転送命令、クリア命令、マルチレジスタ転送命令等を用意した。さらに、物理メモリのメモリ管理やホストマシンとの通信を実現するためにハードウエア資源をアドレス空間上にマッピングしアクセスする命令を用意した。

（２）データ操作命令

データ操作命令は全てレジスタ上のデータに対する操作とし、汎用的なＡＬＵ演算命令の他にタグ操作命令（タグの取り出し、タグの変更）、算術演算命令（整数、浮動小数の加算、減算、乗算、除算）、論理演算命令（AND、OR、EOR、NOT、SHIFT）、整数－浮動小数変換命令等を用意した。これらの命令は高度な最適化を可能とするためにデレファレンス処

Design of CHI Compact Version Architecture
Akihiko KONAGAYA, Ryousei NAKAZAKI, Shinichi HABATA and Mamoru UMEMURA
NEC Corporation

理や出力引数に対するユニフィケーション処理等の論理型言語に特化した機能は備えていない。

（３）実行制御命令

実行制御命令では組込述語の実現に有効なタグ分岐、条件分岐機能を重視し、さらに、高速な分岐を実現するためにdelayed_jump機能を導入した。タグ分岐命令では分岐テーブルをソフトウエアに解放し、任意のタグのグループによる最大８方向までの多方向分岐をソフトウエアで選択可能にした。また、分岐条件としてタグ部の同一比較、整数あるいはアドレスの同一比較、大小比較、浮動小数の同一比較、大小比較等を用意した。さらに、基本命令を用いたプログラムで再帰呼出しを実現できるようにスタック操作命令、サブルーチン呼出し命令を用意した。

４．　組込述語テーブル

ファームウエアで実現する組込述語と基本命令で実現する組込述語を同一のインタフェースで呼出すために組込述語テーブルを用意した。組込述語テーブルに登録した述語は以下の形式の機械語を用いて呼出される。

call_built_in　引数個数，組込述語番号

execute_built_in 引数個数，組込述語番号

このように組込述語をテーブルを介して呼出すことにより組込述語をどのように実現しているかはソフトウエアからは不可視となる。これによりソフトウエアの変更なしに、特定の組込述語のみをファームウエア化することが可能となる。

５．　クローズコンパイル法

最近の言語処理系では、インタプリタは性能よりもデバッグ機能を優先する傾向にあるが、小型化ＣＨＩでは以下の理由からクローズ単位にコンパイルするクローズコンパイル法を採用し、インタプリタの高速実行を図っている。

①論理型言語を用いた知識表現ではインタプリタを動的に変更可能な推論規則、フレーム、データベースの実現手段として使用するため、インタプリタに対しても高速性が要求される。

②クローズをコンパイルすることにより、インタプリタで述語のソースイメージ（クローズ）を取り出す必要がなくなり、冗長なコピーによる性能劣化、メモリの消費を防ぐことができる。

クローズコンパイルした述語の例を図１に示す。各クローズは選択点の生成、戻り先の変更、選択点の除去を行うバックトラック制御命令（ctry_me_else命令、cretry_me_else命令、ctrust_me_else_fail命令）でチェインされる。クローズを追加除去したときは、それぞれ、ctry_me_else命令、 ctrust_me_else_fail命令を書き換えてクローズチェインの張り替えを行う。また、各クローズはゴール呼出しを行う命令列の他にクローズのソースイメージを取り出す命令列を持ち、インタプリタの通常実行では実行命令列を、トレース実行ではソースイメージ取り出し命令列を使用する。

６．　今後の予定

論理型言語を実現するための命令セットに関して詳細設計を完了した。現在さらにインタプリタを高速化するためのインデクシング命令および対象指向プログラミング機能の実現するための命令の詳細仕様を検討中である。

参考文献

[1] Nakazaki,R. etal,Design of a Co-operative High Performance Sequential Inference Machine (CHI), NEC Research & Development, no.80,1986

[2] Warren, D.H., An Abstract Prolog Instruction Set,TR309,Artificial Intelligence Center, SRI International, 1983

ソースプログラム

```
p(X):-q(X),r(a).
p(b).
```

コンパイルコード

```
p/1:  ctry_me_else($1,p/1@1)
      allocate              %実行命令列
      call(q/1)             % p(X):-q(X),
      put_constant(A0,a)    % r(a).
      deallocate
      execute(r/1)              %ソースイメージ
 $1:  get_structure(A0,p/1)      % p(
      unify_variable(A4)         %   X):-
      get_structure(A1,',',/2)   % ','(
      unify_variable(A2)
      unify_variable(A3)
      get_structure(A2,q/1)      % q(
      unify_value(A4)            %   X),
      get_structure(A3,r/1)      % r(
      unify_constant(a)          %   a))
      proceed
p/1@1:ctrust_me_else $1,fail
      get_constant(A0,b)        %実行命令列
      proceed
 $1:  get_structure(A0,p/1)     %ソースイメージ
      unify_constant(b)
      get_constant(A1,true)
      proceed
```

図１　クローズコンパイル例