

逐次型推論マシン：CHI

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

本報告では、通産省の第5世代計算機プロジェクトの一環として、研究開発した逐次型推論マシンCHIの設計思想、高速化手法、性能評価を中心に、小型化版CHIのハードウェア・アーキテクチャについて述べる。

小型化版CHIは、CMOSゲートアレイによるLSI化などにより、デスクサイド型マシンを目指している。タグ・アーキテクチャ、大容量主記憶、PROLOGの為の専用ハードウェア・ユニットを採用し、マイクロ操作の並列処理化を進め、マイクロステップ数を減らすことに努めた。さらに、新しい命令を追加しハードウェアの機能を発揮させるように努めた。この結果、“append”プログラムの実行において、1要素の処理に必要な命令が3命令、マイクロステップ数が12ステップとなり、6MHzのマシン・クロックで500KLIPSの処理速度を得ることができた。さらに、CHIのデータと命令共用のキャッシュメモリをデータ専用と命令専用に分割することで、1要素の処理に必要なマイクロステップ数を8ないし9ステップにできることが分かった。

Co-operative High performance sequential Inference machine : CHI

S.Habata,R.Nakazaki,A.Konagaya,J.Atarashi and M.Umemura

C&C Systems Research Laboratories,NEC Corporation
4-1-1 Miyazaki Miyamae-ku,Kawasaki 213,Japan

(英訳住所)

A Co-operative High performance sequential Inference (CHI) machine has been developed within the Fifth Generation Computer Systems (FGCS) project in Japan. This paper describes the CHI-II (advanced version CHI) hardware architecture, including design approach, its characteristics and performance estimation.

The CHI-II was designed for personal use and to be as small as a desk-side computer, using two CMOS gate-arrays. In the design, tag architecture, large main memory capacity and Prolog oriented specialized hardware were considered for executing micro operations in parallel and reducing the number of micro steps needed in Prolog program execution. New instructions were introduced, in order to take advantage of CHI-II multiple micro operation execution. In this result, 3 machine instructions and 12 micro steps are required for a logical inference in append program. In 6 MHz machine clock, CHI-II performance is estimated to be at 500 KLIPS. Moreover, using data cache and instruction cache, 8 or 9 micro steps are required for a logical inference in append program.

1. はじめに 近年、知識情報処理の研究及び知識情報処理に基づいたシステムの開発が活発である。知識情報処理のプログラム記述言語としては、LISP, PROLOG等がある。このなかで、一階述語論理をベースとしたプログラミング言語は通産省の第5世代計算機プロジェクトの核言語として採用されている。したがって、述語論理に基づいたプログラミング言語で記述される知識情報処理システムが多数出現することが期待できる。

PROLOGなどの述語論理に基づいたプログラミング言語は、優れた記述機能とバックトラックを使用した解の探索機能を備えているため、従来のプログラミング言語より知識情報処理に適している。その反面、言語処理系の負荷が従来言語より大きいため、従来の汎用計算機では十分な開発環境と実行環境を得ることが難しい。そこで、第5世代計算機プロジェクトの一環として、述語論理に基づいたプログラミング言語で記述した知識情報処理システムを高速処理する専用マシン(CHI)の開発を行った。

2. 開発の経緯 CHIは単一ユーザ用の逐次型推論マシンを目標にしている。当プロジェクトでは、CML素子を使用したプロトタイプ(CHI-I)と、TTLとCMOS素子を使用した小型化版(CHI-II)を開発した。CHIハードウェアの特徴は、

- ・タグ・アーキテクチャ
- ・バックエンド型マシン
- ・大容量主記憶
- ・高度に専用化したハードウェア・ユニット

にある。CHIシステム実現に必要な機能である利用者インタフェース及び周辺装置の制御などは、従来の計算機システムの機能を利用することにした。そのため、バックエンド型マシンの構成を採用した。CHIアーキテクチャは、D. H. D. Warrenの提案したWAMアーキテクチャ(1)をもとに、改良を加えたものである。プロトタイプは、CHIアーキテクチャの検証を目的に、1985年に完成し、マシン・クロック10MHzで最高280KLIPS(append実行時)の性能を達成した。(2)この成果を基に、開発したのは小型化版CHI(CHI-II)である。プロトタイプと小型化版CHIの諸元を表1に示す。

3. 小型化版CHIの設計思想 小型化版CHI開発にあたって、以下の目標を設定した。

- (1) デスクサイド型マシン
- (2) 処理速度の向上
- (3) 主記憶容量の拡張
- (4) ホストマシンとの非同期双方向通信
- (5) 汎用命令の導入

項(1)は、プロトタイプは、アーキテクチャの検証、高性能の実現を最優先にして開発した。小型化版CHIは、オフィス等への設置を目指し開発している。項(2)は、尽きることのない知識情報処理研究者の要求に答えるべく、処理速度の向上を目指している。項(3)は、実用的な応用システムを高速実行できる環境を提供す

るため、大きなメモリ空間を半導体メモリで提供することを目指している。項(4)は、ホスト・マシンが備えている周辺装置の機能をCHIから使用できる環境を目指している。第5は、組込み述語の拡張及び修正が容易な環境を目指している。

4. 小型化版CHIのハードウェア技術 小型化し、且つ、主記憶容量を増加し、且つ、処理速度を向上させるため、以下のハードウェア技術を採用している。

- ・CMOSゲート・アレイによるLSI化
- ・64キロ・ビットS-RAM
- ・PAL, PLA
- ・1メガ・ビットD-RAM
- ・サーフェース・マウントによる両面実装基板

プロセッサの制御系を2種類のCMOSゲート・アレイでLSI化している。LSI化した部分のゲート規模は、約2万ゲートで、CHI-Iのプロセッサの約5分に1にあたる。開発したCMOSゲート・アレイの諸元を表2に示す。機械語命令コードの解釈及び、タグ値の解釈は、高速大容量S-RAMで実現し、機械語命令セットの変更及び、タグ値の変更を容易にしている。さらに、高速大容量S-RAMの使用は、IC数の減少、遅延時間短縮の効果もあった。CMOSゲート・アレイによるLSI化が出来なかった制御系の大部分は、高速PAL, PLAを使用し、IC数の減少、遅延時間の短縮を試みている。主記憶容量の拡張は、1メガ・ビットD-RAMの使用とサーフェース・マウントによる両面実装基板の使用で実現している。

5. 小型化版CHIシステムの構成 小型化版CHIシステムは、利用者インタフェース及び、周辺装置の制御を行うホストマシンとCHI-Iマシンで構成する。(図1)ホ

表1 プロトタイプと小型化版CHIの諸元

比較項目	CML版CHI	小型化版CHI
使用素子	CML 256 Kbit D-RAM	TTL, CMOS 1 Mbit D-RAM
基板枚数	81 枚	4 枚
プロセッサ部 主記憶部	約110 枚	18 枚
マシン・クロック	10 MHz	6 MHz
データ幅	36 ビット tag 4または、7 value 32または、29	40 ビット tag 8 value 32
主記憶容量	64 MW (256MB)	128 MW (640MB)
キャッシュメモリ容量	8 KW (32KB)	32 KW (160KB)
マイクロ命令長	80 ビット	78 ビット
制御メモリ容量	11 KW (110KB)	16 KW (約160KB)
"append" 処理速度	280 KLIPS	500 KLIPS
ホストマシン	PSI または、	EWS4800

表2 CMOSゲートアレイの諸元

	MAB	SQB
ゲート数	約6,000 ゲート	約13,000 ゲート
機能	・キャッシュメモリ制御 ・主記憶アクセス制御 ・命令先取りポインタ制御	・マイクロシーケンス制御

ストマシンは、マルチウィンドウ機能を備えている。小型化版CHIの利用者は、ホストマシン上のCHI用ウィンドウを使用し、小型化版CHIをアクセスする。ホストマシンとCHI-IIマシンは標準インタフェースの1つであるSCSIで結合している。CHI-IIマシンは、ホストマシンとの通信制御を行うホスト・インタフェース・ユニット、主記憶ユニットとプロセッサからなる。図2にCHI-IIマシンの構成図を示す。

主記憶ユニットは、論理メモリ空間と物理メモリ空間のマッピングを行うアドレス変換部を含む。図3に小型化版CHIの論理メモリ空間を示す。1つのプロセスは、8個の独立した領域を使用する。その内の4個の領域は、全プロセスに共通の領域となる。論理アドレスは、プロセス番号、領域番号、領域内アドレスの3フィールドからなる。(図4)

6. 処理速度の改善 CML素子からTTL及びCMOS素子への変更は、消費電力、発熱量を少なくしたが、マシン・クロックを下げる原因となった。マシン・クロック低下の問題を解決し、プロトタイプ以上の処理速度を実現するため、以下の点に重点をおいた設計を行った。

- ・メモリ・アクセス速度
- ・マイクロ操作の並列処理
- ・マイクロ分岐の高機能化
- ・ハードウェア機能の効果的利用

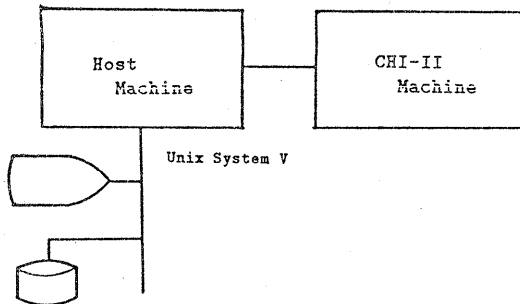


図1 小型化版CHIシステムの構成

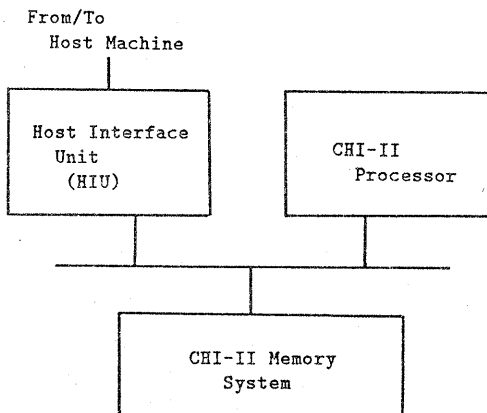


図2 CHI-IIマシンの構成

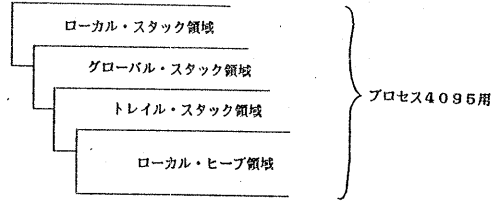
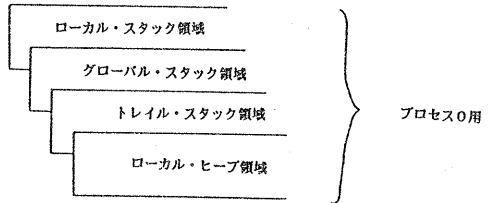
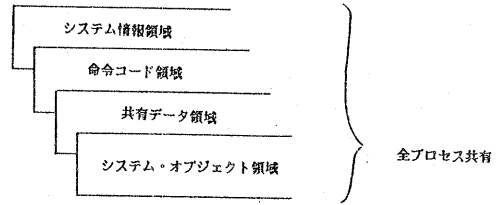


図3 論理メモリ空間の構成

44	32	29	0
プロセス番号	領域番号	領域内アドレス	

図4 論理アドレス・フォーマット

メモリ・アクセス速度 メモリ・アクセスの点で、PROLOGの言語処理系は2つの特徴がある。第1は、PROLOGの言語処理系は、述語読み出し及び、バックトラック処理の為、図5に示すフレームをメモリに生成する。これらのフレームはローカル・スタックと呼ぶ領域に格納する。(1) PROLOGプログラムの実行において、述語呼出の頻度は高く、また、バックトラックはPROLOGの特徴である解探索の基本手段であるから、上記フレームの生成、アクセス、消去の頻度は高い。したがって、PROLOGで記述したプログラムの実行では、従来のプログラミング言語で記述したプログラムの実行の場合より、頻繁にメモリ・アクセスが発生する。第2は、PROLOGの基本処理であるユニフィケーション処理で変数同士の単一化を行った時に、2つの変数を束縛する手段として参照ポインタを使用する。変数間の単一化が複雑に絡み合うと、データをアクセスするために、上記参照ポインタのチェーンを辿る必要が生じる。参照ポインタ・チェーンを辿る処理は、参照ポインタが指すデータを読

BLC :	前選択点フレームへのポインタ
BALT :	次の候補節へのポインタ
BLE :	選択点の環境フレームへのポインタ
BTTOP :	選択点のトレイル・スタック・ポインタの値
BGTOP :	選択点のグローバル・スタック・ポインタの値
BCP :	選択点の述語呼出から復帰番地
BAN :	選択点の引数の個数(値をnとする)
A(0) :	第1引数の値
⋮	⋮
A(n-1) :	第n引数の値

(a) 選択点フレーム

ELE :	前環境フレームへのポインタ
ECP :	述語呼出から復帰番地
BHC :	バンドブック・ハンドラへのポインタ
P(0) :	第1永久変数
⋮	⋮
P(m-1) :	第m永久変数

(b) 環境フレーム

図5 スタック・フレームの構成

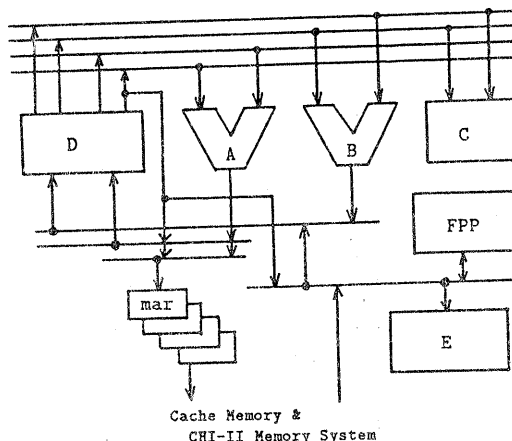
み出し、データタイプを調べる操作からなる。読み出したデータが参照ポインタであれば、さらに、参照ポインタを辿る操作を繰り返すため、メモリ・アクセスをパイプライン処理化して得られる効果が小さい。そこで、メモリ・アクセスをアドレス・セットとデータ・アクセスの2段階に分け、マシン・サイクル時間をキャッシュメモリのアクセス時間から決定した。

マイクロ操作の並列処理 マシン・サイクル時間が決まると、1マシン・サイクル内で、可能な限り多くのマイクロ操作を並列実行させるように努めた。このため、容量が64語のレジスタ・ファイルに4個の読み出しポートと2個の書込ポートを設け、4個のデータ読み出しと2個のデータ書込を1マイクロ・サイクルで処理する。汎用の算術論理演算器の他に、アドレス演算専用の加減算器を用意し、データの演算とアドレスの演算を並列処理する。さらに、メモリ・アドレス・レジスタのインクリメント/デクリメント用加算器、命令先取りアドレス演算用加算器を用意し、データとアドレスの演算の有無に拘らず、メモリ・アドレス・レジスタの更新及び命令先取りアドレスの演算を可能にする。タグ操作機能としては、メモリとレジスタにデータを書込むときに、データのタグを個々に生成することを可能にする。ハードウェア構成設計の目標は、マイクロプログラムの実行時間を、そのマイクロプログラムが行うメモリ(キャッシュメモリ)・アクセスに必要な時間に近づけることである。図6にプロセッサ部の構成を示す。

マイクロ分岐の高機能化 PROLOG言語処理系では、

動的データタイプの検査及び、メモリ・アクセス位置の検査を行う。データタイプの検査機能としては、1データのタグ値検査、2つのデータのタグ値の組合わせの検査がある。1データのタグ値を使用したマイクロプログラム・レベルの多方向分岐機能では、特定方向への分岐を、次の機械語命令の呼出に置き換えている。2つのデータのタグ値の組合わせを使用した多方向分岐では、バリュール部を比較し、一致、不一致の結果も分岐方向の決定に使用する。この場合も、特定方向への分岐を、次の機械語命令の呼出に置き換えることができる。表3にタグ値を使用した多方向分岐の制御の流れを示す。メモリ・アクセス位置の検査は、算術論理演算器とアドレス演算用の加減算器の演算結果を使用する多方向分岐機能である。この時、論理アドレスの領域番号フィールドを比較し、一致、不一致の結果も分岐方向決定に使用する。小型化版CHIが提供するメモリ・アクセス位置検査機能を図7に示す。

ハードウェア機能の効果的利用 小型化版CHIでは、マイクロ操作の並列処理、マイクロ分岐の高機能化を進め、マイクロプログラムの実行に要するマイクロステップ数が減

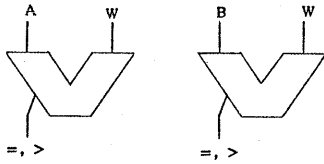


- A : ALU for address calculation
- B : ALU for data computation
- C : argument comparator
- D : multiport register file
- E : instruction prefetcher
- FPP : floating point processor
- mar : memory address registers

図6 プロセッサ部の構成

表3 タグ値を使用した多方向分岐機能

タグ値の組合わせ	バリュール部の比較結果	分岐機能
atom : atom	Xa = Xb Xa != Xb	次の命令呼出 fail処理
nil : nil	---	次の命令呼出
lunb : lunb	Xa > Xb Xa < Xb	変数Xaから変数Xb へのポインタ書込 変数Xbから変数Xa へのポインタ書込



領域番号の関係	領域内アドレスの関係	分岐オフセット	アドレスの関係
Aa = Ba = Va	A <= V, B > V	0	A <= V < B
Aa = Ba = Va	A > V, B <= V	1	B <= V < A
Aa = Ba = Va	A <= V, B <= V	2	A, B <= V
Aa = Ba = Va	A > V, B > V	3	V < A, B
Aa = Ba != Va	-----	4	A, BとWは別領域
Aa = Va != Ba	A <= V, ---	0	A <= V
Aa = Va != Ba	A > V, ---	1	A > V
Aa != Va = Ba	---, B <= V	2	B <= V
Aa != Va = Ba	---, B > V	3	B > V
Aa != Va != Ba	---, ---	4	比較困難

図7 メモリ・アクセス位置検査機能

少するようにハードウェア機能を設計している。特に、マイクロ操作を並列処理するため、多くのデータバス及び演算器を用意している。しかし、WAMで提案された機械語命令セットは、小型化版CHIのハードウェア機能を十分に発揮できない。そこで、インデクシング関係及びリストのユニフィケーション関係の命令を新たに追加している。"append" プログラムを例に、新命令の効果を説明する。図8の(a)がWAMの提案している命令セットで記述した"append" プログラムである。*印の付いている命令が、

? - append([a,b,c],[d,e,f],X) .
 に対して実行される機械語命令である。1要素の処理に、8命令、21マイクロステップを必要とする。図8の(a)の#印の付いている命令

```
switch_on_term $n1,$ls,$fail
$ls: get_list A0
を1命令にした複合命令
switch_on_list
を使用した"append"プログラムが図8の(b)である。第1引数に対するget_list命令が、switch_on_list命令に吸収され、1要素の処理に必要な命令数が7命令、19マイクロステップに減少する。図8の(b)の#印の付いている命令
```

```
get_list A2
unify_temporary_value A3
unify_temporary_variable A2
を1命令にした
get_list_t_val_t_var A2,A3,A2
命令を使用した"append"プログラムが図8の(c)である。第2引数のユニフィケーションのために展開されていた3命令が1命令となり、1要素の処理に必要な命令数が5命令、17マイクロステップに減少する。図8の(c)の#印が付いている命令
```

```
switch_on_list $n1,$ls,$fail
$ls: unify_t_variable A3
を1命令にした
switch_on_list_t_var A3,$n1,$ls,$fail
命令を使用した"append"プログラムが図8の(d)である。第1引数のユニフィケーションを
switch_on_list_t_var A3,$n1,$ls,$fail
```

```
$ls: unify_t_variable A0
の2命令で処理する。1要素の処理に必要な命令数は4命令、15マイクロステップとなる。最後に、図8の(d)で#印が付いている命令
```

```
execute $ap
をインデクシング命令
switch_on_list_t_var A3,$n1,$ls,$fail
```

```
*****
** append([,X,X).
** append([X]A),B,[X]C):-append(A,B,C).
*****
```

```
** $ap: switch_on_term $n1,$ls,$fail
$V0: try_me_else $v1
$n1: get_nil A0
get_t_value A1,A2
proceed
$V1: trust_me_else_fail
** $ls: get_list A0
unify_t_variable A3
get_list A2
unify_t_value A3
unify_t_variable A2
execute $ap
```

(a)

```
* $ap: switch_on_list $n1,$ls,$fail
$V0: try_me_else $2
get_nil A0
$n1: get_t_value A1,A2
proceed
$V1: trust_me_else_fail
get_list A0
* $ls: unify_t_variable A3
unify_t_variable A0
** get_list A2
unify_t_value A3
unify_t_variable A2
execute $ap
```

(b)

```
** $ap: switch_on_list $n1,$ls,$fail
$V0: try_me_else $v1
get_nil A0
$n1: get_t_value A1,A2
proceed
$V1: trust_me_else_fail
get_list A0
** $ls: unify_t_variable A3
unify_t_variable A0
get_list_t_val_t_var A2,A3,A2
execute $ap
```

(c)

```
* $ap: switch_on_list_t_var A3,$n1,$ls,$fail
$V0: try_me_else $v1
get_nil A0
$n1: get_t_value A1,A2
proceed
$V1: trust_me_else_fail
get_list A0
unify_t_variable A3
* $ls: unify_t_variable A0
get_list_t_val_t_var A2,A3,A2
** execute $ap
```

(d)

```
(* $ap: switch_on_list_t_var A3,$n1,$ls,$fail
$V0: try_me_else $v1
get_nil A0
$n1: get_t_value A1,A2
proceed
$V1: trust_me_else_fail
get_list_t_variable A0,A3
* $ls: unify_t_variable A0
get_list_t_val_t_var A2,A3,A2
switch_on_list_t_var $V0
```

(e)

図8 "append"プログラムの展開例

に置き換えることで、1要素の処理に必要な命令数は3命令、12マイクロステップとなる。1要素の処理に必要な命令数、マイクロステップ数、処理速度を、表4に示す。

新たに追加した命令としては、インデクシング命令では、

```
switch_on_list
switch_on_list_t_var
switch_on_list_t_val
switch_on_list_p_var
switch_on_list_p_val
ユニフィケーション用命令では、
get_list_t_var_t_var
get_list_t_val_t_val
get_list_t_var_t_val
get_list_t_val_t_val
```

などがある。新命令追加の為、オペレーション・コードは、10ビットに拡張している。命令コードのフォーマットを図9に示す。

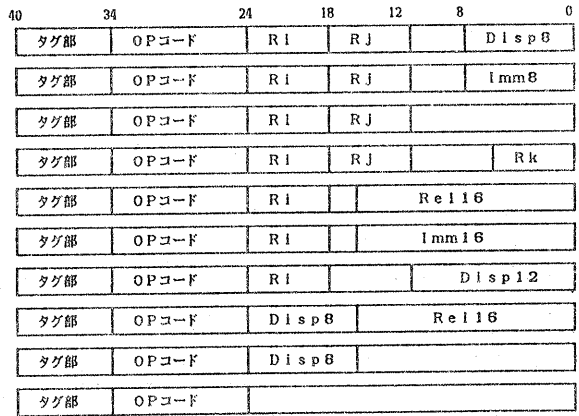


図9 命令コード・フォーマット

7. ホストマシンとの非同期双方向通信 従来システムが備えている機能を流用し、開発を効率化するため、利用者とのインタフェース制御および周辺装置の制御をホストマシンに任せている。しかし、CHI上で応用システムを開発すると、ホストマシン上の周辺装置を使用したり、利用者との会話形式の処理が必要となる。CHI上の応用システムがホストマシンのOS機能を利用するため、非同期双方向通信機能を用意している。小型化版CHIシステムにおけるホストマシンとCHI-11マシンの通信方式を図10に示す。非同期双方向通信機能により、CHI上の応用システムがホストマシンの二次記憶上のファイルにアクセスしたり、キー・ボードからの入力を受けたり、ビットマップ・ディスプレイ上のウィンドウを操作することができる。さらに、CHI-11マシン上で異常が発生した場合は、ホストマシンがCHI-11マシンの状態のダンプ、再立上げの操作を行う。

8. 汎用命令の導入 PROLOGの組込み述語を実現する手段として、汎用命令を用意している。実用的な応用システムをCHI上で開発するためには、組込み述語が提供する算術演算機能や論理演算機能、ベクターの操作機能が必要となる。これらの機能を、状況に応じて、変更したり、拡張する環境を提供するため、大部分の組込み述語は汎用命令のルーチンで実現する。汎用命令の中には、直接、算術論理演算器などのハードウェア・ユニットにマイクロ制御コードを送る命令がある。これにより、本来ソフトウェア・レベルで操作できないハードウェア・ユニットを使用した多方向分岐命令などを実現している。さらに、マイクロプログラムを格納しているコントロール・ストレージの書き換え命令も用意し

表4 小型化版CHIの"append"処理速度

使用命令セット	命令数	マイクロステップ数	処理速度
図9の(a)	8 命令/要素	21 step/要素	285 KLIPS
(b)	7 命令/要素	19 step/要素	315 KLIPS
(c)	5 命令/要素	17 step/要素	352 KLIPS
(d)	4 命令/要素	15 step/要素	400 KLIPS
(e)	3 命令/要素	12 step/要素	500 KLIPS

ており、CHI上のシステムがマイクロプログラムの変更を行うこともできる。

9. 性能評価 機械語命令及び"append"プログラムの実行に必要なマイクロステップ数から、プロトタイプと小型化版CHIの性能評価を行った。評価は、PROLOGのユニフィケーション用命令と"append"プログラムの実行に必要なマイクロステップ数とメモリ・アクセス回数から、マイクロステップ数とメモリ・アクセス回数の比を求め、プロセッサの備えているハードウェア機能とマシン・サイクル時間について、検討してみた。ここで扱うデータは、キャッシュメモリのヒット率を100%として、机上評価したものである。表5は、WAMで提案されたユニフィケーション用命令の実行に必要なマイクロステップ数とメモリ・アクセス回数及びそのデータを基に求めたマイクロステップ数とメモリ・アクセス回数の比である。CHIでは、データと命令がキャッシュメモリを共用している為、メモリ・アクセス回数は、データ・アクセス回数と命令アクセス回数の合計となる。プロトタイプを基にハードウェア構成を改善した小型化版CHIでは、ほとんどの命令の実行が1から2マイクロステップで完了するため、メモリ・アクセスの割合が高いことが分かる。マイクロステップ数減少の結果、表5の命令の平均実行速度に限ると、小型化版CHIはプロトタイプの約1.1倍高速化されている。また、メモリ・アクセスの割合はプロトタイプの約2倍の0.889になっている。表6は、"append"プログラム実行の場合のマイクロステップ数、メモリ・アクセス回数、メモリ・アクセスの割合

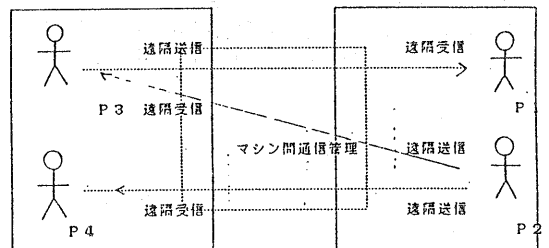


図10 マシン間通信

を示している。小型化版CHIでは、WAMの提案した命令セットを使用した場合でも、メモリ・アクセスの割合は、0.850と高いことが分かる。さらに、新たに追加した命令を使用した場合は、0.917になっている。処理速度では、小型化版CHIはプロトタイプの1.75倍高速化されている。

小型化版CHIのメモリ・アクセスの割合が0.8から0.9ぐらいの値になっていることは、マイクロプログラムの処理時間がマイクロプログラムが行うメモリ・アクセスに要する時間に近づいていることを示している。小型化版CHIでは、データと命令共用のキャッシュメモリを使用している為、命令読み出しとデータ・アクセスを別のマイクロステップで実行する必要がある。この為、“append”プログラムの実行に必要な12ステップのなかに、マイクロ操作の並列処理、命令の先取り処理で省略可能なステップが3ないし4ステップ残っている。

10. まとめ 小型化版CHIはデスクサイド型のパーソナルな逐次型推論マシンを目指し、CMOSゲート・アレイによるLSI化などで小型化を実現した。さらに、処理速度を改善するため、汎用演算器の他に専用の演算器を用意するなどのマイクロ操作の並列処理化を進めた。その結果、処理に必要な時間を、その処理が行うメモリ・アクセス時間に非常に近づけることができた。“append”プログラムの例では、メモリ・アクセス時間と処理時間の比が0.917となっている。したがって、推論マシンのハードウェア・アーキテクチャを検討する上で、メモリ・アクセスを高速化することが非常に重要だとわかる。

CHIでは、キャッシュメモリをデータと命令共用としたが、“append”プログラムの場合、メモリ・アクセスは、データ・アクセス6回と命令アクセス5回からなっており、キャッシュメモリをデータ専用と命令専用に分け、マイクロ操作の並列処理度を高めると、“append”プログラムを実行するために必要なマイクロステップ数は8ないし9ステップにすることができるとわかった。すなわち、マシンサイクル時間が100nSecで、1MLIPS以上のマシンを実現できると考えられる。

謝辞： 最後に、本研究の機会を与えて下さったICOT第4研究室の内田室長、日本電気 C&Cシステム研究所の石黒所長、同所 コンピュータ・システム研究部の山本部長に深謝致します。

参考文献：

- (1) D.H.D.Warren, "An Abstract Prolog Instruction Set", Tech. report 309, Artificial Intelligence Center, SRI International, 1983
- (2) R.Nakazaki, et al. "Design of a High-speed Prolog Machine (HPM)", Proc. of the 12th International Symposium on Computer Architecture, June 1985

Table 5: Memory Access Ratio in typical 18 machine instructions

Machine Instruction	Micro Step Count		Memory Access Count	Memory Access Ratio	
	CHI-I	CHI-II		CHI-I	CHI-II
get_permanent_variable	2	2	2	1.000	1.000
get_temporary_variable	2	1	1	0.500	1.000
get_permanent_value	5	3	2	0.600	0.667
get_temporary_value	3	1	1	0.667	1.000
get_list	2	1	1	0.500	1.000
get_structure	6	3	2	0.500	0.667
put_permanent_variable	3	2	2	0.667	1.000
put_temporary_variable	3	2	2	0.667	1.000
put_permanent_value	4	3	2	0.500	0.667
put_temporary_value	3	1	1	0.333	1.000
put_list	3	1	1	0.333	1.000
put_structure	5	2	2	0.400	1.000
unify_permanent_variable	4	3	3	0.750	1.000
unify_temporary_variable	4	2	2	0.500	1.000
unify_permanent_value	5	3	3	0.600	1.000
unify_temporary_value	5	2	2	0.400	1.000
unify_list	2	1	1	0.500	1.000
unify_structure	6	3	2	0.333	0.667
total	67	6	32	—	—
average	3.7	2.0	1.8	0.478	0.889

Table 6: Performance Estimation in append program execution

	Machine Instruction Count/LI	Micro Step Count/LI	Memory Access Ratio	Performance (KLIPS)
CHI-I	9	35	0.457	285
CHI-II	8	20	0.850	294
without compile optimization				
CHI-II with compile optimization	3	12	0.917	500