記　号　処　理　56－6
計算機アーキテクチャ　84－6
（１９９０．　９．　２８）

記号処理言語プロセッサＯｌｉｖｅのアーキテクチャ評価

幅田　伸一　　丸山　勉　　横田　実
日本電気（株）　Ｃ＆Ｃシステム研究所

記号処理言語プロセッサＯｌｉｖｅのアーキテクチャ評価を行った。Ｏｌｉｖｅは、ＲＩＳＣアプローチによる１チップ化とマシンクロックの高速化に重点を置いたハードウェア・アーキテクチャを採用し、タグ・アーキテクチャ、命令レベル並列処理機能、分岐方向予想型命令の３機能による記号処理言語の高速化を狙っている。さらに、ｌｏａｄ／ｓｔｏｒｅなど汎用命令の高速実行に注意を払っており、従来の記号処理言語専用マシンが苦手としていたＣなどのプログラミング言語も高速実行できる。

評価では、導入した３機能の効果、Ｄｈｒｙｓｔｏｎｅ値による汎用プロセッサとしての処理性能を求め、Ｏｌｉｖｅの有効性を確認すると共に、今後の課題を明確にした。

Performance Evaluation of Symbolic Processing Language Processor: Olive

Shinichi Habata　　Tuyoshi Maruyama　　Minoru Yokota
C&C Systems Research Lab., NEC Corporation
1-1, Miyazaki 4-Chome, Miyamae-Ku, Kawasaki, Kanagawa 213 JAPAN

This paper reports a performance evaluation of an Olive processor, designed to attain high performance in symbolic processing language program execution, especially Lisp and Prolog. Olive is based on a RISC (Reduced Instruction Set Computer), and has three features; a tag-architecture, an instruction-level parallel processing facility and a conditional-branch instruction for squashing.

The effectiveness of three features are measured. Measurement results shows that a tag-architecture attains 1.2 - 1.7 times speed-up, an instruction-level parallel processing architecture improves an execution speed 1.3 - 1.5 times faster, and a conditional-branch instruction for squashing achieves 1.1 - 1.3 times speed-up improvement.

１．はじめに

記号処理言語プロセッサＯｌｉｖｅのアーキテクチャ評価を行った。Ｏｌｉｖｅは、ＲＩＳＣアプローチによる１チップ化とマシンクロックの高速化に重点を置いたハードウェア・アーキテクチャを採用し、タグ・アーキテクチャ、命令レベル並列処理機能、分岐方向予想型命令の３機能による記号処理言語の高速化を狙っている。さらに、ｌｏａｄ／ｓｔｏｒｅなどの汎用命令の高速実行に重点を置き、従来の記号処理言語専用マシンが苦手としていたＣなどのプログラミング言語も高速実行できるプロセッサである。

評価では、高速化の為に導入した３機能の効果、Ｄｈｒｙｓｔｏｎｅ値測定による汎用プロセッサとしての処理性能を求め、Ｏｌｉｖｅの有効性を確認すると共に、今後の課題を明確にした。

２．従来の記号処理言語専用マシンの問題点

記号処理言語はそれまでのプログラミング言語と処理形態が大きく異なり、高速な実行環境を実現するには従来の計算機アーキテクチャと異なる専用アーキテクチャが必要である。従来の記号処理言語専用マシンは、応用システムを全て記号処理言語で記述する単一言語環境を積極的に採り入れていた。この為、記号処理言語以外のプログラミング言語の実行速度が軽視され、ｌｏａｄ／ｓｔｏｒｅなど汎用命令の実行速度が遅い欠点があった。しかし、実用規模の応用システムを分析した結果[3]、ｌｏａｄ／ｓｔｏｒｅなど汎用命令の使用頻度が高いことが判った。

Ｏｌｉｖｅでは上記問題点を考慮し、ＲＩＳＣアプローチによる、１チップ化とマシンクロックの高速化に重点を置いたハードウェア・アーキテクチャを採用した。

また、ソフトウェア面では、コンパイラの最適化処理による冗長な処理の除去が可能となり、記号処理言語の実行速度を遅くしていた特有の処理の頻度を減少できるようになった。このコンパイラの最適化処理などのソフトウェア面での高速化技術を活用する為、Ｏｌｉｖｅでは記号処理言語の処理方式として、最適化コンパイラ／コードジェネレータ方式を採用した。記号処理言語の処理方式は、図１に示す３つの方式に分類できる。処理速度の点では、最適化コンパイラ／コードジェネレータ方式が優れているが、コードサイズの点では、マイクロインタプリタ方式が優れている。３つの方式の利点／欠点をまとめたのが表１である。

３．記号処理言語高速化の為に導入した３機能

Ｏｌｉｖｅでは、タグ・アーキテクチャ、命令レベル並列処理機能、分岐方向予想型命令の３機能による記号処理言語の高速実行を狙っている。

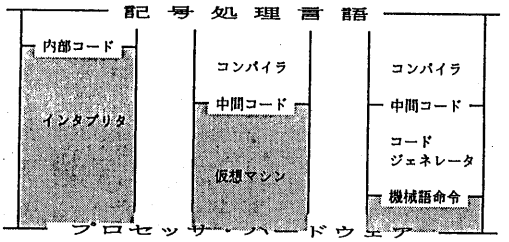

図１．記号処理言語の処理方式

表１．記号処理言語処理方式の比較

処理方式	実行コード	コンパイラによる処理の最適化	最適化率	コード量
インタプリタ方式	内部コード	内部コード生成時のみ	小	小
仮想マシン方式	中間コード	中間コード生成時	中	中
最適化コンパイラ／コードジェネレータ方式	機械語命令	中間コード生成時と機械語命令列生成時	大	大

３．１．タグ・アーキテクチャ

記号処理言語特有の処理を高速化する手段として、タグ・アーキテクチャを採用した。Ｏｌｉｖｅに導入したタグ・アーキテクチャは、タグ値の一致／不一致による条件分岐を行うタグ分岐、タグ値による多方向分岐を行う多方向タグ分岐、データのタグ値を変更してメモリに書き込むタグ書き換えの３処理を高速化する。

タグ分岐処理の高速化

タグ分岐処理用の命令ｊｍｐｔｇを用意した。ｊｍｐｔｇ命令を使用した場合、タグ分岐処理は以下となる。

```
jmptg   arXX, tag_COND, Label
        遅延スロット
```

ｊｍｐｔｇ命令を使用しない場合、Ｏｌｉｖｅでのタグ分岐処理は、タグ・フィールドの切り出し、タグ値の比較、比較結果による条件分岐の３つの処理が必要となる。この場合、以下の様になる。

```
gttg    grØØ, arXX
subi    grØ1, grØØ, tag_COND
bnz     Label
        遅延スロット
```

データはレジスタarXXが保持し、条件のタグ値はtag_COND、分岐先アドレスはLabel、中間結果を保持する作業レジスタとしてgrØØとgrØ1を使用すると仮定している。遅延スロットの影響を無視すると、ｊｍｐｔｇ命令により、タグ分岐処理の処理速度が３倍になる。

タグ分岐処理高速化で期待できる改善度

タグ分岐処理が３倍に高速化されることにより記号処理言語の処理速度改善の期待値ＴＢは以下となる。

$$TB = N/\{N*(1-R_{tb})+N*R_{tb}/3\}$$
$$= 3/(3-2R_{tb})$$

Nはｊｍｐｔｇ命令を使用しない場合の総実行ステップ数、R_{tb}はｊｍｐｔｇ命令を使用しない場合のタグ分岐処理の実行ステップ数が総実行ステップ数に対して占める割合である。

タグ書き換え処理速度の改善度

タグ書き換え処理用の命令ｗｔｔｇを用意した。ｗｔｔｇ命令を使用した場合、タグ書き換え処理は以下となる。

```
wttg   arXX, gr1Ø, tag_j
```

ｗｔｔｇ命令を使用しない場合、Ｏｌｉｖｅでのタグ書き換え処理は、レジスタ上のデータの型を変更する処理と、その結果をメモリに書く処理の２つの操作が必要になる。レジスタ上のデータの型を変更する処理は、タグ・フィールドに即値を書き込むｍｖｔｇ命令を使用する。この場合、以下の様になる。

```
、mvtg   grØØ, arXX, TAG_j
  wt     grØØ, gr1Ø
```

データはレジスタarXX、メモリ・アドレスはdr1Øが保持し、新しいタグ値はtag_j、タグ値変更後の中間結果はgrØØが保持すると仮定している。ｗｔｔｇ命令導入により、タグ書換処理の処理速度が２倍になる。

タグ書き換え処理高速化で期待できる改善度

タグ書き換え処理が２倍に高速化されることにより記号処理言語の処理速度改善の期待値ＴＷは以下となる。

$$TW = N/\{N*(1-R_{tw})+N*R_{tw}/2\} = 2/(2-R_{tw})$$

Nはｗｔｔｇ命令を使用しない場合の総実行ステップ数、R_{tw}はｗｔｔｇ命令を使用しない場合のタグ書き換え処理の実行ステップ数が総実行ステップ数に対して占める割合である。

３．２．命令レベル並列処理機能

処理高速化の基本手段として、命令レベル並列処理機能を導入した。この為、Ｏｌｉｖｅの機械語命令語の形式を以下とした。

主命令　＋　副命令

＋　レジスタのインクリメント／ディクリメント指定

副命令としては、ｍｏｖｅとジャンプ命令が使用できる。命令レベル並列処理機能により、１つの命令で３種の処理の並列実行が可能となった。さらに、タグ分岐命令と後に続くメモリアクセス命令の並列実行を可能とするｒｄｃｋ／ｗｔｃｋ命令を追加した。

命令レベル並列処理機能による処理速度の改善度

ハードウェアの制約から、Ｏｌｉｖｅの機械語命令は１語長とした。この為、機械語命令のビット長が制約となり、並列に実行できる処理の制限が強い。命令レベル並列処理機能には、以下の３使用形態がある。

形態１：　主命令　＋　副命令

形態２：　主命令　＋　レジスタのインクリメント／ディクリメント指定

形態３：　主命令　＋　副命令　＋　レジスタのインクリメント／ディクリメント指定

命令レベル並列処理機能を使用した場合の実行ステップと使用しない場合の実行ステップ数を比較し、命令レベル並列処理機能による処理速度の改善効果を示す。命令レベル並列処理機能を使用していない時の実行ステップ数N_sは以下である。

$$N_s = N + R_c*N + R_r*N + 2*R_{cr}*N$$

Nは命令レベル並列処理機能使用時のステップ数、R_cは形態１の使用頻度、R_rは形態２の使用頻度、R_{cr}は形態３の使用頻度を表す。したがって、平均並列度ＡＰは以下となる。

$$AP = N_s/N$$

命令レベル並列処理機能により処理速度がＡＰ倍になる。

ｒｄｃｋ／ｗｔｃｋ命令による処理速度の改善度

与えられたデータの型を検査し、ポインタである場合、ポインタの指すメモリ番地をアクセスする処理を高速化する為、タグ分岐命令と後に続くメモリアクセス命令を並列実行する必要がある。Ｏｌｉｖｅでは、この処理高速化の為に、ｒｄｃｋ／ｗｔｃｋ命令を用意した。

ポインタが指す先のメモリを読み出す処理は、前記タグ分岐命令ｊｍｐｔｇを使用すると、以下の様になる。

```
!jmptg arXX, TAG_ref, Label_not_ref
    遅延スロット
rd     arXX, arXX
```

！は分岐条件が負論理であることを示し、条件のタグ値とレジスタ上のデータのタグ値が不一致の時、分岐する。データはarXXが保持すると仮定した。TAG_refはポインタのタグ値、Label_not_refはデータがポインタ以外である場合の分岐先アドレスである。ポインタが指す先のメモリへ書き込みを行う処理も同様である。一般に、arXXのデータがポインタである割合が高い為、ｒｄ命令、または、ｗｔ命令をタグ分岐命令と並列実行することで、処理を高速化できる。

ｒｄｃｋとｗｔｃｋ命令は、レジスタが保持するメモリアドレスのタグ値と条件のタグ値を比較し、一致／不一致によりメモリアクセスの実行／中止を制御する。この命令により、上記ポインタが指す先をアクセスする処理に命令レベル並列処理機能が適用可能となる。ｒｄｃｋ命令を使用すると、上記ポインタの指す先を読み出す処理が以下となる。

```
rdck  arXX, arXX & jmptg arXX,
      TAG_ref, Label_not_ref
   遅延スロット
```

ｒｄｃｋ／ｗｔｃｋ命令を主命令、ｊｍｐｔｇ命令を副命令として使用する。遅延スロットの影響がない場合、

ポインタの指すデータをアクセスする処理は１命令で実行できる。したがって、ｒｄｃｋ／ｗｔｃｋ命令により、データの型を調べ、ポインタである場合、ポインタの指す先をアクセスする処理の処理速度が２倍になる。

ｒｄｃｋ／ｗｔｃｋにより期待できる改善度

ｒｄｃｋ／ｗｔｃｋ命令による記号処理言語の処理速度改善の期待値ＲＷは以下となる。

$$RW = N/\{N*(1-R_{rw})+N*R_{rw}/2\} = 2/(2-R_{rw})$$

Ｎはｒｄｃｋ／ｗｔｃｋ命令を使用しない場合の総実行ステップ数、R_{rw}はｒｄｃｋ／ｗｔｃｋ命令を使用しない場合のポインタの指す先をアクセスする処理の実行ステップ数が総実行ステップ数に対して占める割合である。

３．３．分岐方向予想型命令

条件分岐処理を高速化する為、分岐方向予想型命令を導入した。記号処理言語は条件分岐命令の使用頻度が高い。したがって、条件分岐処理の高速化は重要である。分岐方向予想型命令は、条件による分岐発生の有無により、遅延スロットの命令の実行／実行抑止を制御する。この制御により、分岐先の命令を遅延スロットに割り当てることが可能となり、遅延スロットのｎｏｐ命令量を減少し、条件分岐処理を高速化できる。

Ｏｌｉｖｅでは、遅延スロットのｎｏｐ命令頻度が増加すると予想される。第１の理由は、記号処理言語は条件分岐命令の使用頻度が高く、遅延スロットの量が多いからである。第２の理由は、タグ・アーキテクチャと命令レベル並列処理機能による実行ステップ数を減少する過程の中で、条件分岐命令と遅延スロットの命令を１つの命令にする、または、遅延スロットの命令と他の命令を１つの命令にする処理が発生し、遅延スロットに割り当て可能な命令が減少するからである。

遅延スロットのｎｏｐ命令による効果の減少

命令レベル並列処理機能とｒｄｃｋ命令を使用した処理高速化の過程において、遅延スロットのｎｏｐ命令が増加し、処理速度改善の効果が減少する例と分岐方向予想型命令による改善効果について説明する。

```
(car  ar1a ar00)  -----  ①
(car  ar1c ar1b)  -----  ②
```

ar00とar1bはリスト・データへのポインタを、ar1bとar1cは読み出したデータを保持するレジスタである。命令レベル並列処理機能を使用しない場合、以下となる。

```
!jmptg ar00, tag_cons, Label_except1 － ①の処理
nop                                    遅延スロット
!jmptg ar1b, tag_cons, Label_except  － ②の処理
rd    ar1a, ar00        － ①の処理  遅延スロット
rd    ar1b, ar1b                    － ②の処理
```

命令レベル並列処理機能とｒｄｃｋ命令を使用すると、以下の様になる。

```
rdck ar1a, ar00 & !jmptg ar00,
      tag_cons, Label_except － ①の処理
nop                             遅延スロット
rdck ar1b, ar1b & !jmptg ar1b,
      tag_cons, Label_except － ②の処理
nop                             遅延スロット
```

５命令の処理が４命令となる。しかし、２番目の条件分岐命令の遅延スロットもｎｏｐ命令となり、遅延スロットのｎｏｐ命令が１個から２個に増加した。

分岐非優先の分岐方向予想型命令を使用した場合、上記処理は以下となる。

```
rdck ar1a, ar00 & !jmptg ar00,
      tag_cons, Label_except － ①の処理
rdck ar1b, ar1b & !jmptg ar1b,
      tag_cons, Label_except － ②の処理
```

遅延スロットのｎｏｐ命令を全て有効な命令に置き換えることができる。

分岐方向予想型命令による条件分岐処理速度の改善度

分岐方向予想型命令の効果について、従来型条件分岐命令と比較して説明する。従来型条件分岐命令を使用した場合、条件分岐処理の総ステップ数N_sは以下となる。

$$N_s = N_b + (1-R_e)*N_b$$

N_bは実行した条件分岐命令の総数、R_eは実行した遅延スロットの命令が有効命令である確率を表す。右辺第２項は、遅延スロットのｎｏｐ命令により増加したステップ数である。分岐方向予想型命令は、上式第２項の原因となる条件分岐命令を置き換え、遅延スロットのｎｏｐ命令を有効な命令に置き換える。遅延スロットのｎｏｐ命令が減少することにより、条件分岐処理が速くなる。分岐方向予想型命令を使用した場合の条件分岐処理の総ステップ数N_oは以下となる。

$$N_o = N_b + (1-R_{pb})*(1-R_e)*N_b + (1-R_{pe})*R_{pb}*(1-R_e)*N_b$$

R_{pb}は分岐方向予想型命令により遅延スロットのｎｏｐ命令が有効な命令に置き換わる割合、R_{pe}は分岐方向予想が的中する確率をあらわす。第２項は、分岐方向予想型命令を使用しても有効な命令に置き換えができない遅延スロットのｎｏｐ命令のステップ数である。第３項は、分岐方向の予想が外れた時の無効ステップ数である。分岐方向予想型命令は、分岐方向予想が的中した場合、条件分岐を１ステップで実行する。しかし、分岐方向予想が外れた場合、２ステップ必要となる。処理速度の改善度は以下となる。

$$N_s/N_o = \{1+(1-R_e)\} / \{1+(1-R_e)*(1-R_{pe}*R_{pb})\}$$

分岐方向予想型命令により期待できる改善度

分岐方向予想型命令による記号処理言語の処理速度改善の期待値ＢＰは以下となる。

$$BP = N/\{N-R_{pe}*(1-R_e)*N*R_{cb}\}$$
$$= 1/\{1-R_{pe}*R_{cb}*(1-R_e)\}$$

Ｎは分岐方向予想型命令を使用しない場合の総実行ステップ数、R_{cb}は分岐方向予想型命令を使用しない場合の条件分岐命令の実行ステップ数が総実行ステップ数に対して占める割合である。分岐方向予想型命令は、遅延スロットのｎｏｐ命令実行量を減少するもので、遅延スロットのｎｏｐ命令実行量が多い場合に効果を発揮する。したがって、条件分岐命令の使用頻度が高い記号処理言語を処理対象とし、タグ・アーキテクチャと命令レベル並列処理機能により遅延スロットのｎｏｐ命令量が増加するＯｌｉｖｅでは、分岐方向予想型命令の効果が大きくなると期待できる。

４．処理速度改善効果の実測結果

本章では、Ｌｉｓｐ／Ｐｒｏｌｏｇコンテスト、Ｇａｂｒｉｅｌ、ＥＣＲＣの３種のベンチマークプログラムを使用して、各機能の使用頻度を求め、処理速度改善効果を詳細に分析する。

４．１．評価環境

使用頻度の詳細な分析には、ＦＤＬ記述のＯｌｉｖｅシミュレータを使用した。使用頻度とは、総実行ステップ数に対する各機能の実行ステップ数の割合である。分析では、Ｏｌｉｖｅを基準にして、各機能を使用した場合のステップ数と使用しない場合のステップ数を比較し、各機能による処理速度の改善度を求めた。さらに、ベンチマークプログラムの処理時間を汎用マイクロプロセッサ上の商用処理系と比較し、Ｏｌｉｖｅの処理速度を検証した。

４．２．タグ・アーキテクチャによる改善度

タグ・アーキテクチャは、タグ分岐、多方向タグ分岐、タグ書き換えの３処理を高速化する。評価に使用したベンチマークプログラムでは、多方向タグ分岐処理を使用する機会が無かった為、タグ分岐とタグ書き換え処理高速化による処理速度の改善効果を詳細に分析した。

タグ書き換え処理高速化による処理速度の改善効果

評価に使用したＯｌｉｖｅの機械語命令コードでは、タグ書き換え処理を２つの目的で使用している。１つは、ヒープ上にリスト、または、構造体を生成する手段としてである。もう１つは、スタック上に制御フレームを生成する時に、スタックトップなどの制御ポインタのタグ値を書き換える処理である。後者は、ガーベージコレクション処理（以下、ＧＣ処理と略す）などを容易にする為、記号処理言語専用マシンで採用している。この操作は省略が可能である。しかし、タグ書き換え操作の負荷が軽い専用マシンでは、制御フレーム生成時のタグ書き換え操作を行っても、処理速度に影響がない為、ＧＣ処理が容易となるフレーム生成時のタグ書き換え操作を行っている。

後者のタグ書き換え操作を含むコードと取り除いたコードの各々について処理速度改善の効果を分析した。その結果が表２である。Ｌｉｓｐのベンチマークプログラムでは、タグ書き換え命令ｗｔｔｇの使用目的が全て制御フレーム生成時のタグ書き換え操作である。Ｓｔａｋのｗｔｔｇ命令の使用頻度が極端に低い理由は、スタック上のフレームを使用しないからである。

Ｓｔａｋを除く６個のベンチマークプログラムにおけるｗｔｔｇ命令の使用頻度は７～１９％である。結果として、ｗｔｔｇ命令による処理速度の向上は１．Ø７～１．１９倍と、簡単なハードウェア機能の追加で処理速度を改善できることを確認した。

表２．　タグ書き換え処理高速化による処理速度改善の効果

ベンチマークプログラム名	フレームのタグ値設定を含む		フレームのタグ値設定を除く	
	ｗｔｔｇ命令の使用頻度	ｗｔｔｇによる速度改善度	ｗｔｔｇ命令の使用頻度	ｗｔｔｇによる速度改善度
Ｔａｋ	７．３％	１．Ø７倍	Ø．Ø％	１．ØØ倍
Ｔａｋｌ	１５．Ø％	１．１５倍	Ø．Ø％	１．ØØ倍
Ｓｔａｋ	１．３％	１．Ø１倍	Ø．Ø％	１．ØØ倍
Ｑｓｏｒｔ	８．５％	１．Ø８倍	Ø．Ø％	１．ØØ倍
Ｎｒｅｖ	９．３％	１．Ø９倍	Ø．Ø％	１．ØØ倍
ｄｉｆｆ	１Ø．７％	１．１Ø倍	４．４％	１．Ø４倍
ｎｒｅｖ	１８．８％	１．１９倍	１６．１％	１．１６倍

タグ分岐処理高速化による処理速度の改善効果

評価に使用したＯｌｉｖｅの機械語命令コードでは、タグ分岐処理を２つの目的で使用している。１つは与えられたデータのエラー検出である。エラー検出を目的としたタグ分岐処理の場合、分岐の発生は非常に少ない。今回の評価では、エラー検出による分岐は発生していない。もう１つの目的は、与えられたデータの型による処理の選択である。リストデータを処理する場合、リストデータは長さの情報を持っていない為、リストの終端を検査しながら、リストデータを処理する。この場合、リストの終端検出による分岐が必ず発生する。

今回の評価で測定した結果が表３である。Ｌｉｓｐベンチマークプログラムでは、エラー検出を目的としたタグ分岐操作の使用頻度が多い。

Ｔａｋを除く６個のベンチマークプログラムにおけ

表３．　タグ分岐処理高速化による処理速度改善の効果

プログラム名	ｊｍｐｔｇ命令の使用頻度	ｊｍｐｔｇの速度改善度
Ｔａｋ	Ø．Ø％	１．ØØ倍
Ｔａｋｌ	１Ø．４％	１．２Ø倍
Ｓｔａｋ	２１．９％	１．４３倍
Ｑｓｏｒｔ	１８．３％	１．３６倍
Ｎｒｅｖ	１１．８％	１．２３倍
ｄｉｆｆ	１８．４％	１．３６倍
ｎｒｅｖ	２７．３％	１．５４倍

るｊｍｐｔｇ命令の使用頻度は１Ø～２７%であった。結果として、ｊｍｐｔｇ命令により処理速度は１．２Ø～１．５４倍に向上した。

タグ・アーキテクチャによる処理速度の改善効果

タグ・アーキテクチャによる処理速度改善の効果をまとめた結果が表４である。Ｔａｋは整数データのみを使用する為、タグ操作命令の使用頻度が他の６個のベンチマークプログラムより極端に少ない。フレームのタグ値設定処理は省略が可能な処理である。したがって、フレームのタグ値設定処理を除いた場合の効果がタグ・アーキテクチャの効果と考えるべきである。フレームのタグ値設定処理を除いた場合、Ｔａｋを除く６個のベンチマークプログラムにおけるタグ操作命令の使用頻度が１Ø．４～４３．４%であり、処理速度が１．２Ø～１．７Ø倍に向上している。

表４．タグ・アーキテクチャの効果

ベンチマークプログラム名	フレームのタグ値設定を含む		フレームのタグ値設定を除く	
	タグ操作命令の使用頻度	タグ操作命令の速度改善度	タグ操作命令の使用頻度	タグ操作命令の速度改善度
Ｔａｋ	７．３%	１．Ø７倍	Ø．Ø%	１．ØØ倍
Ｔａｋｌ	２５．４%	１．３５倍	１Ø．４%	１．２Ø倍
Ｓｔａｋ	２３．２%	１．４４倍	２１．９%	１．４３倍
Ｑｓｏｒｔ	２７．Ø%	１．４４倍	１８．３%	１．３６倍
Ｎｒｅｖ	２１．Ø%	１．３３倍	１１．８%	１．２３倍
ｄｉｆｆ	２９．１%	１．４６倍	２２．８%	１．４Ø倍
ｎｒｅｖ	４６．１%	１．７３倍	４３．４%	１．７Ø倍

４．３．命令レベル並列処理機能による改善度

命令レベル並列処理機能の処理速度改善効果を、ｒｄｃｋ／ｗｔｃｋ命令を使用しない時と使用する時について分析する。

命令レベル並列処理機能による処理速度の改善度は平均並列度ＡＰ倍となるはずである。しかし、実際の処理速度は総実行ステップ数を基にしている為、キャッシュミスなどの影響を受け、処理速度の改善度は平均並列度ＡＰより低い値となる。

命令レベル並列処理機能による処理速度の改善効果

各使用形態の使用頻度が表５、命令レベル並列処理機能による処理速度の改善度が表６である。ｒｄｃｋ／ｗｔｃｋ命令を使用しない場合、命令レベル並列処理機能の使用頻度が２４．４～４３．９%あり、平均並列度が１．２４～１．４７、処理速度が１．２３～１．４６倍に向上する。ｒｄｃｋ／ｗｔｃｋ命令を使用する場合、ＴａｋとＴａｋｌを除く５個のベンチマークプログラムの平均並列度がØ．Ø３～Ø．１８高くなった。この結果、命令レベル並列処理機能により処理速度が１．２６～１．５４倍に向上する。

理想形の並列度３の命令レベル並列処理機能との比較

Ｏｌｉｖｅで採用した制限の強い命令レベル並列処理機能の効果を、機械語命令のビット長制限を除去し、並列に処理できる操作の制限を取り除いたＬＩＷ（Ｌｏｎｇ Ｉｎｓｔｒｕｃｔｉｏｎ Ｗｏｒｄ）プロセッサの平均並列度と比較した。比較対象としたＬＩＷプロセッサは、図２に示す３個のプロセッサから成り、第１のプロセッサはメモリアクセスと算術演算が可能、第２のプロセッサは算術演算のみ可能、第３のプロセッサは分岐のみ可能とした。比較には、Ｌｉｓｐベンチマークプログラムｑｓｏｒｔを使用した。その結果、上記ＬＩＷプロセッサの平均並列度が１．８７、Ｏｌｉｖｅの平均並列度が１．５６であった。この結果から、ハードウェア、及び、機械語命令のビット長などの強い制限にも拘らずＯｌｉｖｅの命令レベル並列処理機能の効果が大きいことを確認できた。

表５．ｒｄｃｋ／ｗｔｃｋ命令未使用時の命令レベル並列処理機能の使用頻度

プログラム名	形態１の使用頻度	形態２の使用頻度	形態３の使用頻度	命令レベル並列処理の使用頻度
Ｔａｋ	９．８%	１２．１%	２．４%	２４．４%
Ｔａｋｌ	１２．６%	１５．Ø%	４．４%	３２．Ø%
Ｓｔａｋ	２５．７%	１１．４%	Ø．Ø%	３７．１%
Ｑｓｏｒｔ	１１．９%	２Ø．４%	１１．６%	４３．９%
Ｎｒｅｖ	１４．８%	１８．１%	２．９%	３５．８%
ｄｉｆｆ	１２．５%	２１．９%	２．２%	３６．５%
ｎｒｅｖ	９．１%	２Ø．３%	８．６%	３８．Ø%

表６．命令レベル並列処理機能の効果

ベンチマークプログラム名	ｒｄｃｋ命令無し		ｒｄｃｋ／ｗｔｃｋ命令使用頻度	ｒｄｃｋ命令有り	
	平均並列度	速度改善度		平均並列度	速度改善度
Ｔａｋ	１．２７	１．２６	Ø．Ø%	１．２７	１．２６
Ｔａｋｌ	１．３６	１．３６	Ø．Ø%	１．３６	１．３６
Ｓｔａｋ	１．２４	１．２３	１３．５%	１．３７	１．３６
Ｑｓｏｒｔ	１．４７	１．４６	８．７%	１．５６	１．５４
Ｎｒｅｖ	１．３６	１．３６	２．９%	１．３９	１．３９
ｄｉｆｆ	１．３３	１．３２	５．７%	１．３９	１．３８
ｎｒｅｖ	１．２９	１．２９	１７．２%	１．４７	１．４６

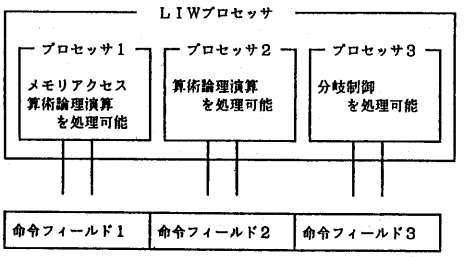

図２．命令レベル並列処理機能の評価に使用したＬＩＷプロセッサ

ＶＬＩＷ化などの命令レベル並列処理機能の拡張により、記号処理の分野においても、さらに高速化が可能であることが判った。しかし、機械語命令のビット長、ハードウェア規模が小さいなどを考慮すると、Ｏｌｉｖｅの命令レベル並列処理機能は強い制限の中で大きな効果を挙げており、有効であることが判る。

４．４．分岐方向予想型命令による処理速度の改善度

分岐方向予想型命令の処理速度改善効果は、遅延スロットがｎｏｐ命令である従来型条件分岐命令の使用頻度に制限される。評価は、遅延スロットがｎｏｐ命令である従来型条件分岐命令の使用頻度を求め、分岐方向予想型命令によるｎｏｐ命令の減少と速度改善効果を分析した。

分岐方向予想型命令による処理速度の改善効果

分岐方向予想型命令による処理速度の改善効果測定の結果が表７である。分岐方向予想型命令の使用頻度が９．９%～３４．８%あり、処理速度が１．０７～１．３３倍に向上した。条件分岐のみの処理速度は１．６０～１．９４倍に向上した。また、分岐方向予想が正しい割合も７５～９７%有り、分岐方向予想型命令による条件分岐処理の速度改善の効果が大きく、有効であることがわかる。

表７．　分岐方向予想型命令の効果

プログラム名	条件分岐命令の使用頻度	分岐方向予想型命令の使用頻度	分岐方向予想が正しい確率	分岐方向予想型命令の効果
Ｔａｋ	２６．９%	９．９%	７５．４%	１．０７倍
Ｔａｋｌ	２４．５%	１１．９%	８４．５%	１．１０倍
Ｓｔａｋ	３３．７%	２７．４%	９２．３%	１．２４倍
Ｑｓｏｒｔ	３０．２%	１７．７%	８７．９%	１．１５倍
Ｎｒｅｖ	２７．２%	１２．３%	９６．９%	１．１２倍
ｄｉｆｆ	３２．７%	２４．４%	８７．２%	１．２０倍
ｎｒｅｖ	３７．５%	３４．８%	９７．０%	１．３３倍

４．５．全体としての処理速度の改善度

タグ・アーキテクチャ、命令レベル並列処理機能、分岐方向予想型命令を組み合わせた時の効果をまとめたのが表８である。使用するデータが整数データのみのＴａｋを除く６個のベンチマークプログラムにおいて、処理速度が１．７～２．６倍に向上した。

処理速度改善の効果が大きいＬｉｓｐベンチマークプログラムＱｓｏｒｔとＰｒｏｌｏｇベンチマークプログラムｎｒｅｖについて、タグ操作命令、命令レベル並列処理機能、分岐方向予想型命令を導入することによる実行ステップ数減少の過程を図３と図４に示す。ＳＴＤは、タグ操作命令、命令レベル並列処理機能、分岐方向予想型命令を使用しない場合の実行ステップ数である。ＴＡＧは、ＳＴＤを基準として、タグ操作命令を導入した場合の実行ステップ数である。ＰＲＡは、ＴＡＧを基準にして、命令レベル並列処理機能を導入した場合の実行ステップ数である。ＰＢＲは、ＰＲＡを基準にして、分岐方向予想型命令を導入した場合の実行ステップ数である。

４．６．商用処理系との処理速度の比較

分析結果を検証する為、ベンチマークプログラムの実行時間を汎用マイクロプロセッサ上の商用処理系と比較した。比較に使用したマシンと言語処理系を表９に示す。汎用ワークステーションの代表として、プロセッサの処理速度とソフトウェア環境の完成度から判断して、ＳＰＡＲＣステーション３７０を選択した。実行時間測定結果が表１０である。

表８．　タグ・アーキテクチャ、命令レベル並列処理機能、分岐方向予想型命令の使用頻度と効果

プログラム名	タグ操作命令の使用頻度	命令レベル並列処理の頻度	分岐方向予想型命令の頻度	処理速度改善度
Ｔａｋ	０．０%	２４．４%	９．９%	１．３倍
Ｔａｋｌ	１０．４%	３２．０%	１１．９%	１．７倍
Ｓｔａｋ	２１．９%	５０．６%	２７．４%	２．０倍
Ｑｓｏｒｔ	１８．３%	５２．６%	１７．７%	２．１倍
Ｎｒｅｖ	１１．８%	３８．７%	１２．３%	１．７倍
ｄｉｆｆ	２２．８%	４２．２%	２４．４%	２．０倍
ｎｒｅｖ	４３．４%	５５．２%	３４．８%	２．５倍

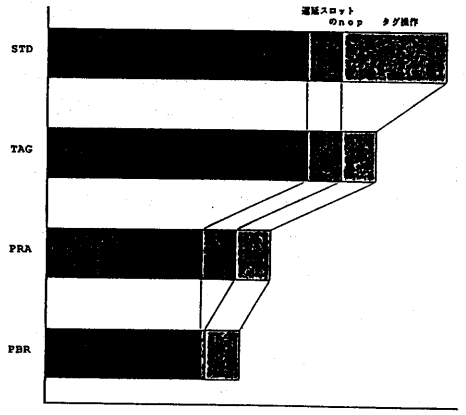

図３．　ＬｉｓｐベンチマークプログラムＱｓｏｒｔにおける実行ステップ数の減少

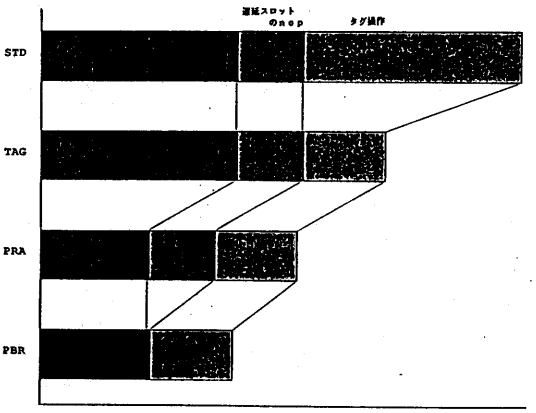

図４．　Ｐｒｏｌｏｇベンチマークプログラムｎｒｅｖにおける実行ステップ数の減少

表９．　測定に使用したシステム

マシン名	マシンサイクル	言語処理系
Ｏｌｉｖｅ	１００ｎＳ	ハンド・コードジェネレーション
ＳＰＡＲＣ	５０ｎＳ	Ａｌｅｇｒｏ ｃｏｍｍｏｎ Ｌｉｓｐ Ｑｕｉｎｕｓ Ｐｒｏｌｏｇ

ＳＰＡＲＣ：　ＳＰＡＲＣステーション３７０

表10．　ベンチマークプログラムの処理速度測定結果

プログラム名	Oliveの処理時間	SPARCの処理時間	対SPARC速度比
Tak	0.14mS	0.11mS	0.8
Tak1	0.95mS	0.46mS	0.5
Stak	0.83mS	1.61mS	1.9
Qsort	1.01mS	2.87mS	2.8
Nrev	1.63mS	6.80mS	4.2
diff	331μS 272KLIPS	670μS 151KLIPS	2.0
nrev	585μS 710KLIPS	1500μS 329KLIPS	2.6

Tak　：　Tak（8、4、2）で測定
Tak1　：　Tak1（8、4、2）で測定
Stak　：　Stak（8、4、2）で測定
Qsort：　50個のデータのソーティング
Nrev　：　30要素のリストのリバース
diff　：　10個のデータのソーティング
nrev　：　30要素のリストのリバース

5．Dhrystone値測定による性能評価

Oliveは、load／storeなど汎用命令の高速実行を重視している。したがって、汎用マイクロプロセッサとしても、充分な処理性能を備えている。汎用マイクロプロセッサとしての処理性能を検証する為、Dhrystone値を測定した。その結果が表11である。Dhrystone値の測定では、MIPS社のワークステーションM120のコンパイラを使用し、コンパイラが出力したMIPSプロセッサの機械語命令列からOliveの機械語命令列を生成した。Dhrystone値測定結果より、同一マシンクロックの場合、OliveプロセッサはSPARCの約1．5倍、MIPSプロセッサの約1．2倍の処理速度を達成できることを確認した。

表11．　Dhrystoneベンチマークプログラム実行結果

システム名	マシンクロック	Dhrystone値
SPARC	50ナノ秒	24、783
MIPS	60ナノ秒	25、424
Olive	100ナノ秒	18、050

MIPS：　MIPS M120

6．考　察

本評価では、Lisp／Prologコンテストのベンチマークプログラム、ECRCのPrologベンチマークプログラムから7つのプログラムを選択し、評価に使用した。これらのベンチマークプログラムは、専用マシン、及び、言語処理系の処理速度評価に広く使用されており、Oliveの処理速度を他の処理系と比較する上で適している。

今回評価に選択した7個のベンチマークプログラムは、Lisp、及び、Prologの特徴的な処理を全て含んでいる。さらに、PrologベンチマークプログラムのWAM命令の使用頻度が、当グループが逐次型推論マシンCHI－Ⅱの評価で使用したDNA配列検索知識ベース実験システムKNOAとほぼ同じである。このことより、今回の評価で得た結果は、応用システムにも当てはまると考えられる。

評価で得た各機能による処理速度改善効果を以下にまとめる。

タグ・アーキテクチャによる
　処理速度向上が、　1．2～1．7倍
命令レベル並列処理機能による
　処理速度向上が、　1．3～1．5倍
分岐方向予想型命令による
　処理速度向上が、　1．1～1．3倍

各機能の処理速度改善効果が低い理由は、以下に示す2点が原因である。第1は、評価に使用したベンチマークプログラムが記号処理言語が提供する特徴的な処理をまんべんなく使用しており、各機能が高速化の対象とする処理の使用頻度が低いことである。第2は、コンパイラの最適化処理により、言語特有の処理が細分化され、冗長な命令の除去、命令の並べ換え操作の中で、ハードウェア機能が高速化の対象とする処理が減少したことである。この2点を考慮すると、上記各機能による処理速度改善度は妥当な値といえる。

7．まとめ

記号処理言語プロセッサOliveを研究開発し、アーキテクチャの評価を行った。評価では、記号処理言語高速化の為に導入した3機能の個々の処理速度改善効果を求め、今後のアーキテクチャ改善の指標を得た。

記号処理言語の高速化では、1つの機能による速度の大幅な改善は期待できず、評価の結果得られた各機能による速度の改善度は大きな効果である。さらに、命令レベル並列処理機能は、VLIW、スーパースカラーなどへの拡張が可能であり、今後、命令レベル並列処理機能を拡張したアーキテクチャの研究を進める必要があることを示している。

さらに、Dhrystone値から、同一マシンクロックに換算した場合、SPARCの約1．5倍、MIPSの約1．2倍の処理速度を達成できることを確認した。

謝辞：　最後に、データ収集に協力戴いた日本電気技術情報システム開発（株）　山岸、河野　両氏と日本電気マイコンテクノロジー（株）　西部氏に感謝致します。

参考文献：

［1］　丸山　他、「AI言語向きRISCアーキテクチャ」情報処理学会 第39回全国大会、1989

［2］ A.Konagaya et al.,"Performance Evaluation of a Sequential Inference Machine CHI," the North American Conference 1989

［3］　横田　他、「LISPマシンLIME」　計算機アーキテクチャ研究会、計算機アーキテクチャ　74－6、　1989．1