# Tachyon Common Lisp の PA-RISC への移植

4D−1

新谷 義弘* 長坂 篤*

*沖電気工業(株) マルチメディア研究所

## 1 はじめに

Tachyon Common Lisp は、RISC マイクロプロセッサ i860$^{TM}$ や SPARC を CPU として持つ UNIX ワークステーション上で稼働している Common Lisp 処理系である[1][2]。

本処理系は、

- 高速な Common Lisp 処理系
- 高い移植性を持つ
- 実用に際して十分な機能を持つ

を特徴としている。

今回、RISC マイクロプロセッサ PA-RISC を CPU として持つ UNIX ワークステーション への移植を行なった。その概要と第2の特徴である移植性の評価について述べる。

## 2 概要

高速性の追求と移植性は矛盾する目標であるが、マシンに依存する部分とマシン独立の部分を明確に分離し、移植の容易さをはかった。高速性に影響する言語の核部分は、マシン独立な上位ライブラリとは分離して、独自の核言語を設定した。

### 2.1 インタプリタの移植

インタプリタは、言語核 Plisp とライブラリに分かれるが、ライブラリは、Lisp で記述されており、移植時には、ほとんど変更なく移植できるので、アセンブラとCで記述された言語核 Plisp が問題となる。

アセンブラで記述されたプログラムは、別の CPU を搭載しているマシンに移植する場合、大幅にコードを書き換えなければならない。

我々は、その負担を軽減するため、強力なマクロ機構を持つアセンブラ処理系を開発している。この新しいアセンブラは、Lisp に似た構文規則を持ち、Common Lisp の持つ強力なマクロ機構と同等なマクロ機構を持つ。CPU の仕様情報を表形式で記述することにより、マシン依存のコードをプログラムから隠蔽している。移植時には、CPU の仕様情報を変更することで移植のほとんどが行なわれる[4]。

### 2.2 コンパイラの移植

一般にコンパイラでは、ソースコードから機械語への変換処理を容易にするために中間言語を設けており、コンパイラは、ソースプログラム⇒中間言語⇒機械語の順に変換する。中間言語を、マシンから独立した仕様にすることで、中間言語への変換までの処理は一般にマシンから独立になっている。これに対して、最適化を含むコード生成部は、通常、マシンおよび OS に依存したものとなる。

## 3 PA-RISC への移植の問題点

前節で述べたように、移植にともなう作業の多くは、それぞれのＣＰＵの命令セットにいかに対応するかである。

３つのＣＰＵの違いを簡単に述べると、

| | i860 | SPARC | PA-RISC |
|---|---|---|---|
| 条件フラグ | 1種類 | 複数種類 | 複数種類 |
| 条件分岐 | 26 bit | 22 bit | 12 bit |
| コール | 26 bit | 30 bit | 17 bit |
| 即値命令 | 16 bit | 13 bit | 最長11 bit |
| データ開始番地 | コードに続く | コードに続く | #x40000000 以降 |

となり、i860 と SPARC は、ほぼ似たような命令体系であるが、PA-RISC は、全く異なる。

### 3.1 分岐

PA-RISC の分岐命令は、条件分岐、相対アドレス分岐、絶対アドレス分岐の３種類である。

PA-RISC では、条件分岐のアドレスフィールドが１２ bit、相対アドレス分岐のアドレスフィールドが１７ bit であり、Common Lisp のような多くの関数群を持ち、しかも、関数を再定義可能な言語では、アドレスがかなり離れる場合があり、関数間の呼び出しを条件分岐や相対アドレス分岐で行なうことができない。従って、全て２命令かかる絶対アドレス分岐を使用することにした。

### 3.2 同一関数内の条件分岐

同一関数内において、条件分岐命令のアドレスフィールドが１２ bit しかないと、分岐できない可能性がで

Porting of Tachyon Common Lisp to PA-RISC
Yoshihiro Shintani* and Atsushi Nagasaka*
*Oki Electric Industry Co, Ltd., Media Laboratories

てくる。従って、相対アドレス分岐を併用する必要がある。しかし、分岐に1命令余分にかかることになるので、アセンブラの最適化において条件分岐ができない距離の場合についての処理をすることにした。

### 3.3 即値命令

演算命令では、即値フィールドを扱えるのは加算と減算だけであり、それ以外の演算は、一度レジスタに代入してから、演算を行なう必要がある。

### 3.4 データセグメント

PA-RISC では、データセグメントがかなり高位番地から始まっている。Tachyon Common Lisp では、下位数ビットをタグとして使用しているため上位数ビットが消えてしまう。よって、データアクセスには上位数ビットを補完する必要がある。

### 3.5 スペース

PA-RISC は、アドレスの上位2ビットでスペースを切り分けている。他のスペースにあるプログラムに分岐する場合は、そのスペースを指定する必要がある。

HP-UX では、テキスト領域とデータ領域は違うスペースにあり、ユーザが定義する関数などをコンパイルしたコードはデータ・スペースに存在するため、テキスト・スペースにある関数とのやりとりは、スペースの指定を行なう必要がある。呼び出す方はどのスペースの関数を呼び出すかあらかじめわかっているが、呼ばれた側はどのスペースから呼び出されたかを調べて、それぞれのスペースを指定して戻る必要がある。

### 3.6 Nullification

PA-RISC では、ある命令の結果がある条件を満足すれば、次の命令を無視するものである。うまく利用すれば、高速化につながる。

例えば、リストの検査は、これまでの処理では、コンスタグでない場合一度分岐し、その後で nil かどうかの検査を行なっていたが、

```
; *input0*=入力レジスタ0,*temp*=テンポラリレジスタ,
; *nil*=nil値レジスタ
EXTRU    *input0*,30,2,*temp*
                        ; タグのとりだし
SUBI,=   #b11,*temp*,%r0
                        ; コンスタグなら次命令をスキップ
COMBF,=  *nil*,*input0*,label-not-cons
                        ; nil でなければエラーへ
```

となり、nil の場合の処理が速くなった。

## 4 移植性評価

本処理系は、大規模な処理系である。移植の概要を以下に示す。

| | 行数 (行) | 移植工数(人月) | |
|---|---|---|---|
| | | SPARC | PA-RISC |
| インタプリタ | 約 110000 | | |
| アセンブラ | 約 60000 | 2.0 | 2.5 |
| C | 約 6000 | | |
| Lisp | 約 45000 | | |
| コンパイラ (lisp) | 約 37000 | 0.5 | 0.75 |

移植期間は、PA-RISC への移植の方が若干多くかかった。これは、i860 と SPARC に比べて、PA-RISC のアーキテクチャが大きく変わっていることに原因がある。しかしながら、移植は、約15万行のプログラムを3.25人月という短期間で行なうことができた。これは、

- インタプリタ
  システム依存部を除きアセンブラが、トランスレータの変更だけでできたこと。
  実際に、かかった移植工数のほとんどがこのトランスレータの移植であった。
- コンパイラ
  CPU 依存部の分離により、多くのものが簡単な置き換え操作でできたこと

による。

## 5 おわりに

Tachyon Common Lisp の PA-RISC プロセッサへの移植について報告した。

今後、改善の余地があるインタプリタやコンパイラの最適化についてさらに研究、評価を行う予定である。

## 参考文献


[1] 五味他: "Tachyon Common Lisp の実現方式", 記号処理研究会 92-SYM-64-3, 情報処理学会, 1992

[2] 長坂他: "Tachyon Common Lisp: An Efficient and Portable Implementation of CLtL2", *Proceedings* of the 1992 ACM Conference on Lisp and Functional Programming, ACM, 1992.

[3] 新谷他: "Tachyon Common Lisp の SPARC への移植", 情報処理学会第45回全国大会,4Q-4

[4] 山田他: "Tachyon Common Lisp: 表駆動アセンブラ・トランスレータによるPA−RISCへの移植", 情報処理学会第47回全国大会,4D-2