Tachyon Common Lisp の SPARC への移植

4Q−4

新谷 義弘* 長坂 篤* 高橋 順一** 五味 弘**

*沖電気工業(株) 総合システム研究所 **(株)沖テクノシステムズ ラボラトリ

1 はじめに

Tachyon Common Lisp は、Common Lisp 第2版[1]に基づいた処理系であり、RISC マイクロプロセッサ i860 を CPU として持つ UNIX ワークステーション OKIstation7300 上で稼働している[2][4]。

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。第1の特徴である高速性については、すでに報告済み[5][6] であるので、本論文では、第2の特徴である移植性について、RISC マイクロプロセッサ SPARC を CPU として持つ UNIX ワークステーション SPARCStation2 への移植を行なった結果をもとにその概要と評価について述べる。

2 処理系概要

本処理系の構成を図1 に示す。インタプリタは、言語の核部分である Plisp とその上位の Common Lisp ライブラリ関数からなる。コンパイラは、Lcode と呼ぶ中間言語への変換部と Lcode からオブジェクトコードを生成するコード生成部および対象マシンの記述ファイルからなる。

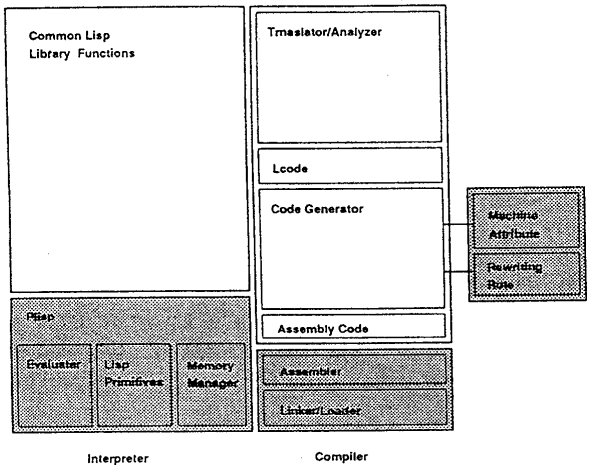

図1 Tachyon Common Lisp 処理系の構成

Porting of Tachyon Common Lisp to SPARC
Yoshihiro Shintani*, Atsushi Nagasaka*
Junichi Takahashi** and Hiroshi Gomi**
*Oki Electric Industry Co, Ltd., Systems Laboratories
**Oki Technosystems Laboratory, Inc.

2.1 インタプリタ

インタプリタの核部分は Plisp と呼ぶマシン独立な仕様を持つ言語であり、ライブラリは Plisp およびそれから作られる Common Lisp で記述されている。

Plisp は高速性を重視するため、アセンブリ言語とC言語で記述されているが、Lisp ライクなマクロアセンブラ [3]で記述することにより他のプロセッサへの移植を容易にしている。

2.2 コンパイラ

本処理系を他機種へ容易に移植できるようにするために、各 CPU のハードウェア属性(レジスタの構成／用途、各命令の仕様等)を記述したハードウェア属性ファイルを CPU ごとに用意する。コンパイラは中間言語をアセンブリ言語に変換する際に、ハードウェア属性ファイルを参照し、そこから得られる CPU の情報から対象となる CPU のオブジェクトコードを生成する。

これによりコンパイラを新しい CPU に対応させることが容易に行なえる。コンパイラ自体は、Common Lisp で記述されている。

3 処理系の移植

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。

3.1 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[3]。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 依存部の大部分をマクロ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、マクロを変更することで移植のほとんどが行なわれる [7]。

3.2 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。本処理系では、マシン独立な中間言語Lcodeを採用している。Lcodeへの変換部はマシン独立であるが、コード生成部の移植性を高くするために、次のような構成にしている。

- マシン属性の記述とコード生成部の分離
  マシン属性の記述をコード生成部外に持たせることにより、コンパイラ自身のプログラムの変更を行なわないでマシンの変更が可能となる。
  尚、マシン属性には、
  ・レジスタの個数　・レジスタの用途
  ・条件分岐の種類　・各命令の仕様
  ・制御レジスタ　　・スタック仕様
  等が記述されている。
- マシン依存のコード生成/最適化の変換ルール
  マシンによって最適化できるコード、特に、インライン展開するための情報をマシンごとの変換ルールとして記述したものである。
  これらの情報もコード生成部より分離することで、別のマシンへの変更がコンパイラ自身のプログラムの変更を行なわないで、変換ルールのデータベースを切り替えることで行なえる。

4 評価

4.1 移植性評価

本処理系は、大規模な処理系である。

SPARCへの移植は、約15万行のプログラムを2.5人月という短期間で行なうことができた。

	行数	移植工数
インタプリタ	約 100000 行	2.0 人月
アセンブラ	約 50000 行	
C	約 6000 行	
Lisp	約 45000 行	
コンパイラ(lisp)	約 37000 行	0.5 人月

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

によると思われる。

4.2 性能評価

i860とSPARCの2つのCPUの場合の本処理系の実行速度を表1に記す。

この2つのCPUの整数演算性能(SPECint)は、ほぼ同じである。

総合的に見て、SPARCの方が若干速い。これは、移植によって性能が悪くなっていないことを示しており、性能の面においても、Tachyon Common Lispの移植性が高いことを実証した。

尚、コンパイルされたTakrの速度差は、インストラクションキャッシュの容量の差と思われる。

表1　ベンチマーク関数の実行速度(単位:　秒)

	i860		*SPARC*	
	Interpreter	Compiler	Interpreter	Compiler
Tak	1.850	0.046	1.761	0.041
Stak	3.120	0.243	2.915	0.255
Ctak	2.307	0.165	2.981	0.179
Takl	13.270	0.115	15.031	0.123
Takr	1.894	0.133	1.183	0.052
Boyer	26.730	0.770	25.260	0.710
Browse	40.060	0.455	31.380	0.405
Destruct	7.650	0.197	6.975	0.166
Init-traverse	55.650	1.378	54.380	1.162
Run-traverse	263.480	3.849	233.800	3.504
Derivative	4.220	0.280	3.430	0.290
Dderivative	5.220	0.390	4.040	0.330
Div2-iter	5.960	0.157	5.340	0.163
Div2-rec	4.610	0.140	4.180	0.147
Fft	2.320	0.142	1.880	0.128
Puzzle	39.050	1.546	34.130	1.278
Triangle	514.790	17.746	432.380	14.042

5 おわりに

Tachyon Common Lisp のSPARC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のアルゴリズム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。SPARCには、レジスタウィンドウがあるが、現在、本処理系では使用していない。関数呼出しの多いLisp処理系における、レジスタウィンドウの有効利用法もそのひとつである。

また、ANSI Common Lispへの対応も行う計画である。

参考文献

[1] Guy L. Steele Jr.：“Common Lisp the Language Second Edition” ,Digital Press, 1990

[2] 大江他 ：“OKI Common Lisp の開発 -概要-”, 情報処理学会第43回全国大会,5L-2

[3] 高橋他：“OKI Common Lisp におけるアセンブラプログラムの開発環境”, 情報処理学会第43回全国大会,1J-7

[4] 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

[5] 新谷他：“Tachyon Common Lisp コンパイラの高速化方式”, 記号処理研究会 92-SYM-64-4,情報処理学会,1992

[6] 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.

[7] 高橋他：“Tachyon Common Lisp におけるアセンブラ・トランスレータの移植性”, 情報処理学会第45回全国大会,4Q-5