

LISPマシン LIME のソフト開発 (II)

2P-5

— コンパイル技法 —

越前 孝 越前章子 富田一則 南田英輝
神戸日本電気ソフトウェア株式会社

1. はじめに

LISPマシン LIMEのハードウェアは、通産省第五世代計算機プロジェクトの成果を利用して試作されたものである。その母体となったPROLOGマシン CHI-II[1]は、PROLOGの高速実行に要求されるメモリ操作、引数レジスタ、複数のスタックを実現するための論理アドレス方式、メモリの動的割り当て等を備えている。そしてそれはそのままLISPの高速実行のためにも必須の機能であり、LISPマシンとしても充分有効である。本稿では、これらの特徴を活かしたLISP処理方式とそれを有効に利用するためのコンパイル方式およびその評価等について述べる。

2. LISP処理方式

基本的には、汎用機上でのUTILISPと同様の処理方式であるが、ここでは汎用機上でのUTILISP処理方式と異なる点について述べる。

(1) 名前空間

多重プロセス環境の導入により、グローバル空間とローカル空間に分かれており、システム変数のようなシステムおよび関数実体は、グローバル空間に置かれている。[2]

(2) スタック

関数の呼び出しや戻りのための実行順序制御情報の格納と、呼び出された関数に含まれるローカル変数の永久変数領域に使用されるコントロールスタックと、引数のラムダ束縛のために使用されるバインディングスタックの2本が用いられる。

(3) 引数レジスタ方式

LIMEのハードウェアは、PROLOGマシンとして設計されており、引数受け渡しに用いられるようなハードウェアスタックを装備していない。従って、LIMEでもPROLOGにおけるWAM方式と同様に、コンパイルされた関数においては、引数レジスタを用いた引数受け渡し方式を採用した(インタプリタでは、引数は直接バインディングスタックを介して受け渡される)[3]。

また、PROC構文内の一時変数のように、仮引数以外で用いられるローカル変数も、原則として引数レジスタにとられ、退避の必要が生じた時点でのみ、引数レジスタとともにコントロールスタックへ退避される(永久変数)。

LIMEでは、引数レジスタを32個持っているが、コンパイラがそのうち4個を特定の目的に使うので、28個が実際の引数受け渡しに使える。通常は、これぐらいの数で充分であるが、実用上は、ローカル変数の数が28個を上回る場合も出てくる。一時変数がレジスタに割り付けられないときは、コントロールスタックの変数領域に割り付け、引数のみで28個を上回る場合は、コンパイル時の制限事項とした。これは、たとえ28個をレジスタに割り付ける方式をとったとしても、省略可能引数の考慮や作業用レジスタの不足を考慮しなければならず、コンパイル処理を煩雑にするだけで、効果は大きくないと判断したためである。どうしても29個以上の引数を引き渡したい場合には、構造体として渡すこととした。ただし、不定個引数をとるシステムの組込関数には29個以上の引数を、29個目からはコントロールスタック経由という具合に引き渡せるようにした。

3. コンパイラ

(1) コンパイラの構成

本コンパイラは、下記のような3つのパスから構成される。

- ①関数定義のリスト構造(構文木)をたどって、LISP構文としてのインターフェイス、マクロ定義の展開、構文の同等な別表現への置換(LIST関数をCONS関数で表現するなど)を行う。併せて、各構文木における変数の使用状況や関数の呼び出し状況など、構文木のグローバルな情報を収集し、各構文木内のノードへ保持させる(PASS-1)。
- ②前パスで収集されたグローバルな情報を、前方の構文木に伝える(PASS-2)。
- ③各構文木のノードに保持した、後方における変数使用状況等を含んだグローバルな情報と、種々の管理テーブルをもとに、ソースに対応したLIME機械語の命令列を生成し、その後、不用命令の削除、分岐命令の最適化等、局所的最適化を行う(PASS-3)。

(2) コンパイルの方式

引数レジスタ方式であることから、できるだけレジスタ上の値を有効に利用するということと、コントロールスタックへの退避を最小限に抑えたコードを生成することを、本コンパイル機能の主眼とした。

①グローバル情報の利用

複数引数をもつ関数のある引数の評価結果に関して、その後の引数評価において関数呼び出しがあればレジスタが破壊される。この場合、その破壊の時点で、以前の引数の評価結果レジスタを退避することも可能であるが、その時点に至るまで無駄にレジスタを使用していることになる。これを防ぐために、関数呼び出し状況保持したグローバル情報を参照する。引数評価した時点で、この情報からその後に関数呼び出しがあると分かれば、結果レジスタを退避しておきレジスタを解放しておく。

```
(FOO (CAR A) (CDR B) (FEE C) (CAR D))
      評価結果を退避      評価結果はレジスタ
```

また、関数呼び出し時のレジスタ退避処理においては、変数使用状況保持したグローバル情報を参照し、関数呼び出し後も使用される有意義な変数(レジスタ)のみをコントロールスタックに退避するようにする。

```
(CONS (FOO (CAR A) (FEE B) B C) ...)
```

FEEの呼び出し時点ではB Cのみ退避

②同値関係テーブルの利用

変数とレジスタおよびスタック三者の同値関係を保持するテーブルを、同値関係テーブルと名付ける。この同値関係テーブルは、変数の値がレジスタやスタック上へ移動する都度、必ずその内容の同値性を維持させる。そして変数アクセスの際には、まず同値なレジスタの有無を参照することにより、レジスタの内容を有効に利用する。

また、CONDなどの分岐構文に関しても、この同値関係テーブルを分散使用し、合流地点では分岐先からの複数の同値関係テーブルの共通部分をとることにより、レジスタを有効に利用する(図1)。

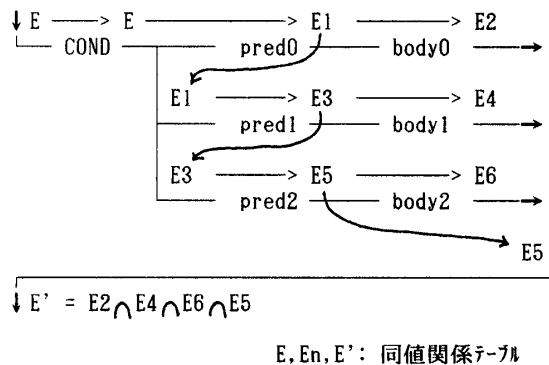


図1 分岐構文における同値関係テーブル使用

4. ローカル関数

実行性能を、さらに向上させるためにローカル関数機能を導入した。通常、LISPにおいては、シンドロームで関数呼び出しが行われており、それによって関数呼び出しの柔軟性が実現されている。しかし、この柔軟性を犠牲にすれば、FORTRAN またはC のようなコンパイラ言語のオブジェクトと同様に、直接的なアドレス参照で、関数呼び出しを実現することもできる。このような、直接的なアドレス参照解決される関数をローカル関数と呼び、その実現を図った。

ローカル関数の宣言は次のような、コンパイラへの指示文で行う。

```
(DECLARE (FOO FEE) LOCAL)
```

第一引数に指定された関数名(リスト)が、ローカル関数として取り扱われる。ローカル関数宣言が有効なのは、同一ソースファイル内のみである。このような機能を導入した場合、当然、この有効範囲内の関数を集めて、ローカル関数呼び出しのアドレス参照の解決を行うリンクフェーズが必要となる。それらをまとめて一つに構成したものをモジュールとした。(モジュールがLIMEの実行単位である)

LIME上のUTILISPシステムはLISPで記述されており[2]、実行効率を上げるためにこのローカル関数を活用して実現されている。

以下に、第三回LISPコンテスト[4]の課題の中から、サブルーチン呼び出しの存在するものを選んで、それらをローカル関数宣言した場合と、そうでない場合のコンパイルコードの実行性能測定結果を示す。

表1 ローカル関数機能適用の効果

コンパイルコード	ローカル 宣言 あり	ローカル 宣言 なし
LIST-TARAI-4	194	208
SREV-5	4.2	4.4
SREV-6	16.8	18.4
QSORT-50	3.5	3.6
NREV-30	5.4	6.2

ローカル関数機能を適用すれば、10% 弱の性能向上がみられる。しかし、それ程大きく効果が出なかったのは、LISP専用機であるLIMEにおいては、関数呼び出しのための処理が高速で、全処理中で占める割合が、さほど大きくなかったことに起因すると思われる。

5. コンパイルコードの評価

第三回LISPコンテスト[4]関数をコンパイルし実行した結果を以下に示す。いずれもGC時間を含んでいない。

表2 LISPコンテスト 実行結果 (単位: ms)

コンパイルコード	LIME UTILISP
TARAI-4	53
TAK-18-12-6	268
LIST-TARAI-4	208
SREV-5	4.4
QSORT-50	3.6
NREV-30	6.2
STRING-TARAI-4	431
FLONUM-TARAI-4	247
SEQ-100	12

ただし、LIMEのクロックは200ns で測定した。

以上のように、他のLISP処理系[5]と比較しても遜色のない高速性を持っていることが分かる。また、LIMEインタプリタの性能評価[2]と比べれば、コンパイルすることにより、6 ~ 38倍の性能向上が見られる。特に関数呼び出しを評価するプログラムにおいて性能向上が顕著である。これは関数呼び出しを専用命令化し、専用機としての効果を出したことにもよるが、インタプリタをLISPで記述したことによる特徴が大きく出ているものと考えられる。

6. まとめ

当初、LISPマシン LIMEは、16MHz 版のEWS4800 をホストプロセッサとし、その3 ~ 5 倍の性能を目標として設計された。今回の評価により当初の目標は達せられた。しかし、LISPのようにアーキテクチャの低い言語に対しては、現在および今後の汎用プロセッサの性能向上により専用機との差は確実に縮まっていくものと思われる。従って、今後はLISP専用機の実現方法も含めて、そのメリット/デメリットを慎重に検討していかなければならないであろう。

参考文献

- [1] 幅田 他: 逐次型推論マシンCHI小型化版のハードウェア, 情報学会第33年全国大会, B5-8, 1986.
- [2] 越前 他: LISPマシンLIME のソフト開発 (I) -LISP 記述LISPシステム-, 情報学会第38年全国大会, 1989.
- [3] 横田, 越前 他: LISPマシン LIME. 情報学会計算機7-77研究資料, 1989.
- [4] 奥乃: 第三回Lispコンテスト 及び第一回Prologコンテスト の課題案, 情報学会, 記号処理28-4, 1984.
- [5] 奥乃: 第三回Lispコンテスト と第一回Prologコンテスト 報告, 情報学会, 記号処理33-4, 1985.