QuickHandy インタープリタ言語における高速化の実現

出雲正尚

神奈川大学理学部情報科学科

QuickHandy は UNIX の X-Window 上で動作し、コンピュータグラフィックスや複素数まで含めた数値計算を容易に行えるように作成されたインタープリタ言語である。高速な演算機能を実現しているので、複雑な計算を行いながら、その計算結果をウィンドウに図として示していくことができる。QuickHandy の文法は C 言語に似ているので，C 言語プログラマーはあまり多くの文法を覚える必要がなく，容易にプログラムを組むことができる．本稿では QuickHandy について簡単に述べ、さらにその高速化のための内部機構について述べる。

Implementation of rapid processing method in QuickHandy interpreter language

Masanao Izumo

Department of Infomation Science, Faculty of Science, Kanagawa University

QuickHandy is an interpreter language for computer graphics on X-Window and for numerical calculation including complex numbers. It is high speed interpreter to process the operations. So processing many complex probrems, it can show the results as the figure on the window. The syntax is like C language, therefore C programers need not learn some syntax of QuickHandy, and can make the program easily. This paper briefly explans about the syntax, and describes the internal processing method to run the program efficiently.

1 はじめに

Lisp のシンタックスシュガーで QuickHandy 言語に似た形のシステムがあるが、実行速度が遅く、システムが大きくなって利点が失われる。軽くて高速で使いやすいプログラミング言語にグラフィックス機能をつける目的で QuickHandy を作成した。

QuickHandy 言語の主な特徴は以下のようなものがあげられる。

- 構文が C[1] 言語に似ているので C 言語を知っている人は容易にプログラムが組める。
- 型なしの言語で、 Lisp のように変数にはどんなデータも代入できる。
- インタープリタ言語である。
- グラフィックスが容易に行える。
- 複素数計算まで含めた数値計算が高速に行える。

X-Window 上で C 言語を用いてグラフィックスプログラミングを行うには Xlib[2] を用いて図を描画しなければならない。しかしながら、 Xlib は非常に低レベルなライブラリであり、扱いが困難である。また、グラフィックスプログラミングにおいては、コンパイラ言語を用いるより、インタープリタ言語を用いた方が、有用なことが多い。なぜなら、グラフィックスにおいては、色、座標、などのパラメタが頻繁に変更されることがあるからである。

また、数値計算において、楕円関数を用いる計算[3] などには多くの複素数計算が必要とされる。さらに、その計算結果が図としてあらわされる場合は、グラフィックスの機能が必要である。

QuickHandy の言語形式としては、 C 言語の文法に似たインタープリタを採用している。これは、 C 言語プログラマーにとって、新たに覚える文法が少なくてすむ為と、インタープリタであれば、グラフィックスが容易であると考えた為である。

現在の QuickHandy のグラフィックス機能は点や線などを描く為の、基本的なものしか実現されていない。より高度なグラフィックス機能、ボタンやメニューなどのウィンドウは今のところ実現されていない。

2 プログラム例

本節では QuickHandy プログラムの例を簡単に示す。

以下のプログラムはマンデルブロート集合を描画するプログラム例である。複素数でマンデルブロートの関数を定義できるので、簡潔にプログラムが組める。 QuickHandy の文法は非常に C 言語に近い。

```
// プログラム 1
#define WIN_WIDTH 200
#define WIN_HEIGHT 200
#define MINX -2.0
#define MINY -1.5
#define MAXX 1.0
#define MAXY 1.5
#define DX   ((MAXX - MINX) / WIN_WIDTH)
#define DY   ((MAXY - MINY) / WIN_HEIGHT)
#define N 32            // 最大反復回数
f(z, r) = z * z + r;    // 文関数定義
gopen(WIN_WIDTH, WIN_HEIGHT);
```

```
for(x = MINX, wx = 0; wx < WIN_WIDTH; x += DX, wx++) {
    for(y = MINY, wy = 0; wy < WIN_WIDTH; y += DY, wy++) {
        z = 0, r = complex(x, y);
        for(i = 0; i < N; i++) {
            z = f(z, r);
            if(abs(z) > 2) break;
        }
        if(i == N) gpset(wx, wy);
    }
}
gpause();
```

次に QuickHandy でのクイックソートの関数定義を示す。

```
// プログラム 2
function qsort(a, first, last)
{
    local i = first, j = last;
    local t, x = a[(first + last) / 2];

    for(;;) {
        while(a[i] < x) i++;
        while(x < a[j]) j--;
        if(i >= j) break;
        t = a[i], a[i] = a[j], a[j] = t;
        i++, j--;
    }
    if(first < i - 1) qsort(a, first, i - 1);
    if(j + 1 < last)  qsort(a, j + 1, last);
}
```

3 インタープリタの高速化

インタープリタを高速に動作させるには、与えられたソースプログラムをそのまま実行するのではなく、実行速度が上がるように、中間形式に変換し、その変換されたコードを実行する必要がある。ただし、インタープリタなので、時間がかかる変換はできない。中間形式の構造は様々なものが考えられるが、主な中間形式[4]として、以下の様なものが考えられる。

- 後置記法
- 構文木
- 四つ組 (三番地コード)

どの中間形式で実行するかは、言語仕様によって様々な形が考えられる。QuickHandy は四つ組を採用している。これは、QuickHandy の言語仕様から考えて、最も効率的に処理できると考えたからである。

4 コード変換の流れ

QuickHandy のソースコードを四つ組に変換して実行するまでの大まかな過程は以下の図のようになる。

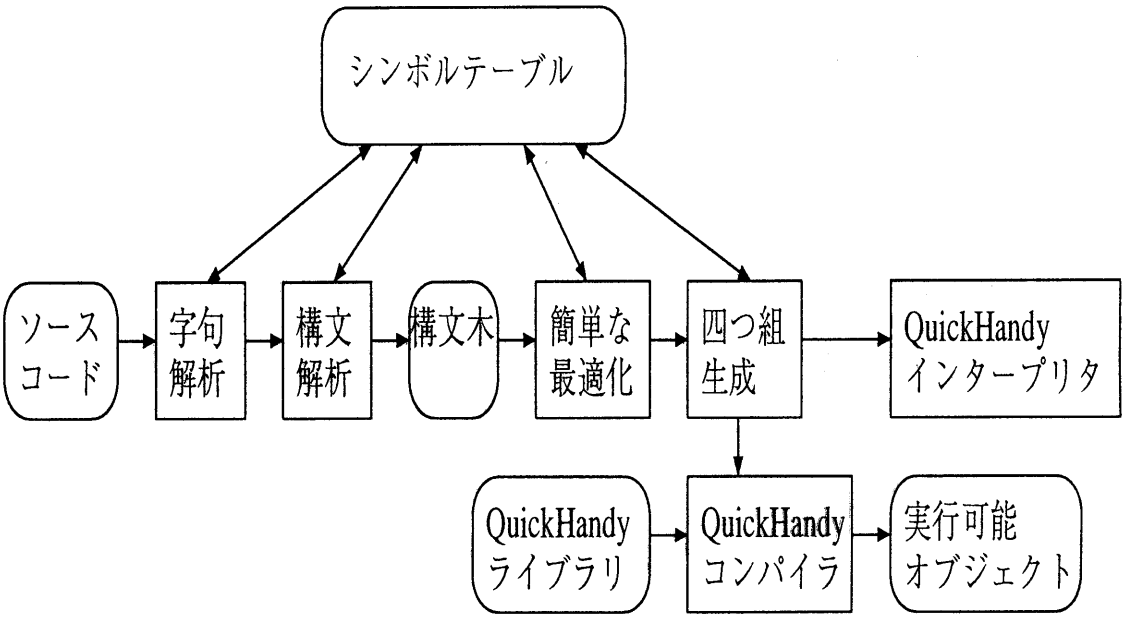

図 1: コード変換フェーズ

ここでまるで囲まれたところはデータを表し、四角で囲まれたところは処理を表す。

変換中は、シンボルテーブルを作成し、名前に関する情報を記録していく。実行時には、このシンボルテーブルは使用しない。四つ組生成中に、データにはすべて、整数値であるアドレスが割り振られ、実行中はこのアドレスを用いてデータをアクセスする。そのため、変数の値の参照や代入が高速に行なうことができる。

式は、一度、構文木に変換されてから、四つ組に変換するが、それ以外の文は構文解析後、直接、四つ組に変換する。

なお、プログラムを実行可能オブジェクトにするための QuickHandy のコンパイル機能もある。

5 四つ組のインタープリット

5.1 コード変換例

ソースプログラムが四つ組みに変換されたコードの例として、0 から 10 までの和を求めるプログラムを示す。プログラムは

```
for(sum = i = 0; i <= 10; i++)
    sum = sum + i;
```

となる (表示部分は除く)。このプログラムは以下のような四つ組みに変換されてから実行される。

```
1     [6]  assign     M[38]     M[23]     -
1     [7]  assign     M[37]     M[38]     -
      [8]  goto       11        -         -
2     [9]  add        M[37]     M[37]     M[38]
1     [10] xinc       M[39]     M[38]     -
1     [11] iflegoto   9         M[38]     M[22]
```

左端の数字はソースプログラム上での行番号であり、次に命令番号、命令コード、アドレスが続く。M[ ] の中の数字はメモリ中へのアドレスを表し - は 未定義を表す。M[23] には 0、M[22] には 10、が実行前の初期化の段階で代入されている。

5.2 基本的な動作

命令コードの解釈は、命令コードを添字とする、関数へのポインタの配列を用いて実現している。例えば、実際の add 命令は、命令コード 260 であり、260 番目の関数 qh_add() を呼び出すことで四つ組を実行していく (図 2)。この命令コードは yacc が生成したトークンの記号定数をそのまま用いている。命令を実行し終えたら、プログラムカウンタ (PC) を 1 つ増やし、次の命令を実行する。

QuickHandy のアクセス可能な全てのデータには、「型」を表すタグと、そのデータの「値」がある。各命令は与えられたデータの型をチェックし、その型に応じた処理が行なわれる。

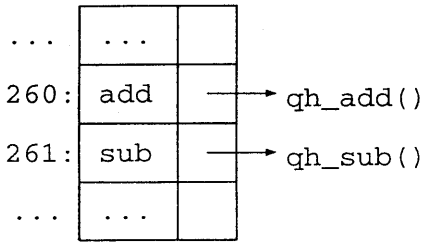

図 2: 命令コードとその処理関数との対応表

5.3 関数呼び出し

QuickHandy の関数呼び出し時において、ローカル変数を実現するために、データが格納されているメモリはスタック構造となっている（実行時スタック）。関数が呼び出されると、スタックポインタ (SP) が適切な位置に設定される。どれだけ SP を移動するかは四つ組生成時に決定し、実行時には既に求まっている。

関数呼び出しの動作は以下の順に行なわれる。

1. 実行時スタックに引数をスタックにプッシュ
2. 呼び出し前の SP, PC、および、関数の返り値が入るアドレスの保存（別のスタックにプッシュ）
3. PC, SP の更新

1. は引数をプッシュするための複数の四つ組に展開される。2. と 3. は 1 つの命令でまとめて実行される。

関数から戻る時は、SP, PC を復元し、関数の返り値を設定する。

6 ガーベージコレクション

次に QuickHandy に用いられているガーベージコレクションの機構について説明する。不要なメモリは以下の様なコードを実行すると発生する。

```
function foo()
{
    local x = new int[10];      // ローカル配列の生成
//    ....
    return;
}
```

foo を呼び出すと、foo 中で配列が生成され、その配列はローカル変数 x に代入される。ここで生成された配列は関数 foo() から抜けると二度と参照されなくなる。QuickHandy は、このようなデータを再利用するためのガーベージコレクションの機能を実現している。

QuickHandy の配列データは、実体へのポインタで表現されている。実行前に全ての配列データをフリーリストでつないでおく。new が呼ばれると、フリーリストから配列データを取ってくる。フリーリストが空になるとガーベージコレクションが発生する。

ガーベージコレクションの最初のステップは、実行時スタックの中にあるすべての配列データにマークをする。次に、マークのない配列データをフリーリストに戻すことにより、未使用の配列データを再利用可能にする。

7 実行時間

以下の表に、実際の問題を解くのにどれくらい時間がかかるかを示す。マシンは SPARK station 5(S-4/5) 上の SunOS 4.1.3-JL で行なった。

プログラム	実時間 (秒)	ユーザ時間 (秒)
プログラム 1(マンデルブロート集合の描画,200 × 200 画素)	10.62	5.27
プログラム 2(クイックソート, データ数 10 万)	20.72	20.50
ガウスの消去方 (軸選択法, 100 元)	4.99	4.91
エラトステネスのふるい (10 万までの素数)	3.25	3.22
行列の積 (100 × 100)	17.02	16.84
$f(x) = \sum_{k=1}^{1000} \frac{2}{k\pi} \sin \frac{k\pi}{500} x, \quad (x=1, 2, \ldots, 1000)$	16.74	16.50

8 今後の課題

今後の課題について以下のものがあげられる。

- 新たなデータ型の導入 (構造体、連結リスト型、多倍長整数型など)
- ウィンドウ機能の高度化
- 文字列処理の効率化、正規表現
- 強力なデバッグツール、トレース機能
- より高度な最適化機能の実現
- 可変長引数関数の定義
- エラー処理モジュール

参考文献

1) B.W. カーニハン、D.M. リッチー、石田晴久訳: プログラミング言語 C、第2版、共立出版株式会社 (1989)。

2) Adrian Nye: Xlib Reference Manual 3rd Edition, Vol. 2、O'Reilly & Associates, Inc. (1992)。

3) Eiichi Goto, Masanao Izumo, Yamazaki Yoshiyiro: Flow in Packing Lattice of Circular Holes (Fuluid Dynamics Research に投稿予定)。

4) A.V. エイホ、R. セシィ、J.D. ウルマン 共著、原田賢一訳：コンパイラ I、II 原理・技法・ツール、サイエンス社 (1990)。

5) John R. Levine, Tony Mason & Doug Grown: lex & yacc, 2nd Edition、O'Reilly & Associates, Inc. (1990)。