CbC による Perl6 処理系

135730B 氏名 清水隆博 指導教員：河野 真治

Abstract

In Rakudo Perl6 itself written in a subset of Perl6 called NQP (NotQuitPerl). NQP Bytecode is interpreted by MoarVM wrriten in C language.

We are developing Programing Language that name Continuation based C (CbC). CbC has Code Gear and Data Gear as programing language units. A transfer from a Code Gear to another Code Gear is performed using a CbC's goto statement, which is compiled as a jump instruction in CbC. MoarVM is going to implemented in C language, it can be rewritten in CbC. In this thesis, consider rewriting the instruction dispatch part of MoarVM with CbC.

We successfully MoarVM writtern in CbClang. CbC is faster than Original MoarVM has CPU cash.

1 Perl6 の現在の実装

現在開発が進んでいるプログラミング言語 Perl6 は、 入力されたソースコードを複数の仮想機械で実行可能なバイトコードにコンパイルする。仮想機械は Perl6 専用の VM である、 MoarVM と、 JVM が選択可能である。MoarVM で動作する主流な Perl6 の実装に、 Rakudo がある。

Rakudo 実装の Perl6 は、 他スクリプト言語と比較すると、 実行速度が低速である。また、 MoarVM の実装が巨大な case 文が使用されていたりとモジュール化がし辛い状況にある。バイトコードから仮想機械を動作させる際、 バイトコード中のバイト列から、 実行する命令の処理を取得するフェッチ部分が時間的にはボトルネックとなっている。

2 CbC

Perl6 処理系の改良には、 gcc と LLVM/Clang 上に実装した、 Continuation based C(CbC) を用いる。CbC は関数よりも細かな単位である、 CodeGear を基本的な処理単位とし、 CodeGear の遷移でプログラムを記述する C の下位言語である。CodeGear の宣言は、__code という型を持つ関数として宣言する。__code は内部的には void 型として扱っているが、 プログラマからの扱いは CodeGear である事を示す指示子のような役割である。CodeGear は C の関数とは異なり返り値を持たず、 呼び差し元の関数に戻る代わりに別の CodeGear へ遷移する。CbC における CodeGear 間の継続は、 スタックに値を積まず、 環境も遷移時に持たない為軽量継続と呼ぶ。CbC は軽量継続を中心にプログラミングする事が可能であるため、 レジスタの移動などが行われない。その為並列化やループ制御、 関数呼び出しにおけるスタック制御などを意識した、 プログラミングスタイルでプログラミングする事が可能である。

```
__code cg1(TEST testin){
    TEST testout;
    testout.number = testin.number + 1;
    testout.string = "Hello";
    goto cg2(testout);
}

__code cg2(TEST testin){
    printf("number␣=␣%d\t␣string=␣%s\n",testin.number,
         testin.string);
}

int main(){
    TEST test = {0,0};
    goto cg1(test);
}
```

ソースコード 1: cbc_example.cbc

3 MoarVM

MoarVM は Perl6 の専用の処理系であり、 C 言語で実装されている。MoarVM は Perl6 及び、Perl6 の実装に用いられている、Perl6 のサブセット言語である NQP が生成したバイトコードを解釈する。バイトコードは、 NQP のインタプリタから出力可能である。

JIT コンパイルなどが現在導入されているが、 起動時間などが低速である問題がある。

MoarVM はレジスタマシンであり、 レジスタ操作や数値の加算を行う命令が存在する。この命令は、 Perl6 プログラムが変換されたバイトコードによって指定される。

4 MoarVM のディスパッチ

MoarVM は NQP から変換されたバイトコードを読み取り、 都度実行する。MoarVM の場合この処理は MVM_interp_run という関数で行われている。この関数内では MoarVM が実行すべき命令が並んだ命令列を持ち、その値で巨大な case 文、 または C のラベルジャンプによって分岐させる。分岐後は命令に応じた処理を MoarVM が行い、 次の命令を実行する。分岐後は命令にごとの処理を実行し、 現在の命令列から、 実行した命令の長さ分命令列を進める。進めた後に case 文の先頭もしくは C のラベルジャンプを利用して次の処理に遷移する。この一連の処理を命令コードのディスパッチと呼ぶ。命令コードの実行の中で

は、 現在の MoarVM のレジスタである reg_base やスレッドごとの環境である構造体 tc などを参照する。この方法の問題点として、 巨大な case 文に変換されてしまう事から、命令に対応する処理のモジュール化が出来ない。ディスパッチ部分の処理が都度走る為に、 低速になる。ラベルに break point を設定できない為に、 デバッグし辛いなどが挙げられる。

```
DISPATCH(NEXT_OP) {
    OP(no_op):
        goto NEXT;
    OP(const_i32):
        MVM_exception_throw_adhoc(tc, "const_iX␣NYI");
    OP(const_i64):
        GET_REG(cur_op, 0).i64 = MVM_BC_get_I64(cur_op, 2)
            ;
        cur_op += 10;
        goto NEXT;
}
```

ソースコード 2: MoarVM 内のインタプリタのディスパッチ

5 CbCMoarVM のディスパッチ

本研究では MoarVM は 2018.04.01 のバージョンで実装している。命令コードの実行すべき単位は、CbC の CodeGear の単位として扱える為、 命令処理を CodeGear に変換する。変換された CodeGear は、 オリジナルの MoarVM の命令コードと対応させる為に、 CodeGear の配列に格納する。MoarVM はこの配列を参照し、 要素として得られる CodeGear に軽量継続を行う。CodeGear での処理が終了すると、次の CodeGear を決定する為に必要な計算を cbc_next という CodeGear で行い，次の命令列に軽量継続する。

```
__code cbc_no_op(INTERP i){
    goto cbc_next(i);
}
__code cbc_const_i32(INTERP i){
    MVM_exception_throw_adhoc(i->tc, "const_iX␣NYI");
   goto cbc_const_i64(i);
}
__code cbc_const_i64(INTERP i){
    GET_REG(i->cur_op, 0,i).i64 = MVM_BC_get_I64(i->
        cur_op, 2);
    i->cur_op += 10;
    goto cbc_next(i);
}
```

ソースコード 3: CbCMoarVM 内のインタプリタの状態遷移

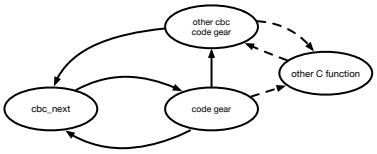

図 1: CbC における MoarVM バイトコードインタプリタ内の状態遷移

オリジナルの MoarVM とは異なり、同一関数上で実行する訳では無い。従って、 MVM_interp_run で定義している局所変数のレジスタ reg_base などにアクセスすることは通常では出来ない。CodeGear の遷移において、 これら必要なインタプリタの情報を纏めた構造体 INTER を宣言し、このポインタである INTERP を引数として入力する。各 CodeGear ではこの INTERP を経由することでレジスタ情報などにアクセスする。

この方法の利点として CodeGear は単なる関数として扱える為、元のソースコードの様に同一ファイル内に全ての処理を書く必要がない。またラベルには break point を設定する事が出来なかったが、 CodeGear はデバッガからは関数として見る事ができるため通常の C の関数の様に break point を設定する事が可能である。

6 まとめ

現在は MoarVM を利用し NQP, Perl6 のビルドが達成出来た。またテストコードも、 元の MoarVM と同様の達成率を得た。処理速度は再帰呼び出しなどを行っている例題では、 オリジナルの MoarVM より低速であるが、単純ループなど、 命令が CPU のキャッシュに残る場合は高速に動く事が測定された。

7 今後の課題

本研究では、 MoarVM の命令ディスパッチ部分が CodeGear の集合に変換可能である事を示した。また、MoarVM の命令ディスパッチ部分を CodeGear の集合に変換するスクリプトや、 CbC で動作する MoarVM を作成した。今後は速度向上を目的とし、 ディスパッチ時に配列集合にアクセスせず、直接次の CodeGear に遷移する Threaded Code を CbC を用いての実装を検討する。また、 スクリプトが、使用する変数代入や分岐までの単位である、 基本ブロックを判別し、 動的に CbC のコードを生成する symbolic execution などの実装を検討する。

参考文献

[1] 徳森 海斗, 河野真治. Llvm clang 上の continuation based c コンパイラ の改良, 2015.

[2] 伊波立樹, 東恩納琢偉, 河野真治. Code gear、data gear に基づく os のプロトタイプ, 2016.

[3] The LLVM Compiler Infrastructure. http://llvm.org.