

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 written in C language.

We are developing Programming Language that name Continuation based C (CbC). CbC has Code Gear and Data Gear as programming 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 writtern MoarVM in CbC. When an instruction executed by CbC enters the CPU cache, it runs faster than the original MoarVM.

1 Perl6のVMのモジュール化

現在開発が進んでいるプログラミング言語にPerl6がある。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。コンパイルされたバイトコードは、Perl6専用のVMであるMoarVMやJavaVMが評価する。MoarVMで動作する主流なPerl6の実装に、Rakudoがある。

Rakudo そのものはPerl6のサブセットであるNQPで実装されている。RakudoやNQPが動作するMoarVMはC言語で実装されている。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必要がある。この部分の実装が巨大なcase文やラベルgotoが使用されており、モジュール化が行われていない。モジュール化を行うことで、それぞれに対して最適化を行うことが可能となる。このモジュール化では、Cの関数より小さな単位であるCbCが適していると考えられる。その為、本研究では、MoarVMの命令対応部分のモジュール化の検討を行う。

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、呼び出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\tstring=%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。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

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

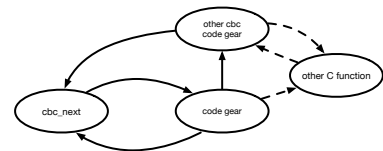


図 1: CbC における MoarVM バイトコードインタプリタ内の状態遷移

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、単純ループなどの例題の場合、命令が CPU のキャッシュに残る場合は 1.5sec 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ことを示した。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] Perl6 documentation. <https://docs.perl6.org/>.
- [2] 徳森海斗, 河野真治. Llvm clang 上の continuation based c コンパイラの改良. 琉球大学工学部情報工学科平成 27 年度学位論文 (修士), 2015.