Continuation based C の LLVM/clang 3.5 上の実装について

徳 森 海 斗† 河 野 真 治†

当研究室では並列・分散プログラミングスタイルとして Data Segment, Code Segment を用いるプログラミング手法を提案している．この手法を用いるプログラミング言語として CbC の開発を行っており，これは C の下位の言語になる．本研究では，LLVM/clang-3.5 をベースとした CbC コンパイラの実装を行い，LLVM/clang-3.5 への CbC の具体的な実装について述べる．

The implementation of Continuation based C Compiler on LLVM/clang 3.5

Kaito Tokumori† and Shinji Kono†

We suggest a programming paradigm which use data segments and code segments. We develop CbC which is a lower language of C and uses that programming paradigm. In this study, we implement CbC compiler on LLVM/clang and introduce implemented Continuation based C Compiler on LLVM/clang-3.5.

1. 研究目的

当研究室では，プログラムをコードセグメント，データセグメントという単位を用いて書くという手法を提案している．この手法を用いてプログラミングを行う言語として Continuation based C (以下 CbC) というプログラミング言語を開発しており，これは C の下位の言語にあたる．CbC においてコードセグメント間の処理の移動は goto 文を用いた軽量継続によって行われ，これは Tail Call Elimination というコンパイラの持つ最適化の強制によって実現される．CbC では継続前の code segment に戻ることはなく，状態遷移ベースのプログラミングを行うのに適しており，これは OpenCL, CUDA, そして Cerium といった並列開発環境を用いたプログラムの記述に向いている．

これまでに開発された CbC のコンパイラは Micro-C をベースにしたものと GCC をベースにしたものの二種がある． GCC 上に CbC コンパイラを実装した理由の一つに，当時の UNIX 環境における コンパイラの標準が GCC であったからというものがあった．しかし，Mac OS X の最新版である Mavericks では GCC の代わりに LLVM/clang が用いられるようになり，環境が変わりつつあることがわかる．このような背景から，LLVM/clang を用いて CbC をコンパイルできるのが良いという考えが生じた．本研究では LLVM/clang 上に CbC コンパイラの実装を行う．

† 琉球大学
University of the Ryukyu

2. Continuation based C (CbC)

CbC では C の関数の代わりに code segment を用いて処理を記述し，code segment 間の移動に goto を用いる．構文は C と同じであるが，ループ制御や関数コールが取り除かれる．

code segment の記述は C の関数の構文と同じで，型に __code を使うことで宣言でき，code segment 間の移動は goto の後に code segment 名と引数を並べて記述することで行える．この goto による処理の遷移を継続と呼ぶ．図 1 は code segment 間の処理の流れを表している．

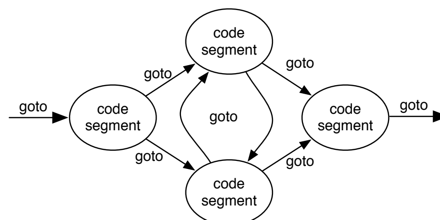

図 1 goto による code segment 間の継続

code segment は C の関数と異なり戻り値を持たず，処理が終われば次の code segment へと処理を移る．C において関数呼び出しを繰り返し行う場合，呼び出

された関数の引数の数だけスタックに値が積まれていく．しかし, 戻り値を持たない code segment ではスタックに値を積んでいく必要が無く, スタックは変更されない．このようなスタックに値を積まない継続, つまり呼び出し元の環境を持たない継続を軽量継続と呼び, 軽量継続により並列化, ループ制御, 関数コールとスタックの操作を意識した最適化がソースコードレベルで行えるようになる．

以下の図 2 に示されたプログラムは与えられた数値の階乗を算出する CbC プログラムである．

```
__code print_factorial(int prod)
{
  printf("factorial = %d\n",prod);
  exit(0);
}
__code factorial0(int prod, int x)
{
  if ( x >= 1) {
    goto factorial0(prod*x, x-1);
  }else{
    goto print_factorial(prod);
  }
}
__code factorial(int x)
{
  goto factorial0(1, x);
}
int main(int argc, char **argv)
{
  int i;
  i = atoi(argv[1]);

  goto factorial(i);
}
```

図 2 階乗を計算する CbC プログラムの例

3. LLVM/clang で扱われる内部表現

CbC コンパイラの実装の前に, LLVM 及び clang で扱われる内部表現について触れる．LLVM, clang はコンパイル対象コードを clang Abstract Syntax Tree (clangAST), LLVM IR, Selection Directed Acycric Graph (SelectionDAG), Machine Code, MCLayer の順に変換し，その後アセンブラ言語へと変換する．図 3 は clang がソースコードを読み込み, アセンブラ言語を出力するまでの流れを表した図である．

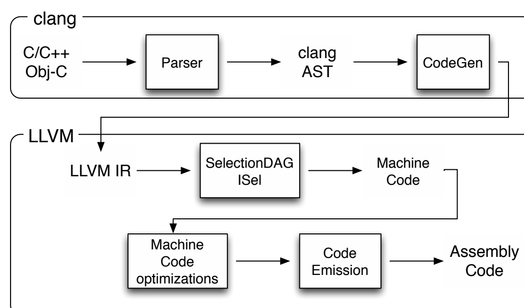

図 3 clang, LLVM によるコンパイルの一連の流れ