Continuation based C コンパイラのGCC-4.6による実装

学籍番号:085711E 氏名:大城信康 指導教員：河野真治

H23 11/18 fri

1 研究背景と目的

当研究室ではプログラムをコードセグメント (Code Segment) 単位で記述するプログラミング言語 Continuation based C (以下 CbC) を開発している。また CbC の開発と共に CbC のコンパイラの開発も行なってきた。2008 年には GCC-4.2 をベースとした CbC のコンパイラ (以下 CbC-GCC) が開発され、GCC の最適化やデバッグ、他アーキテクチャへの対応と言った恩恵を受けられるようになった。以降、GCC のアップデートに合わせて CbC-GCC のアップデートも行ってきた。

本研究では、GCC-4.5 をベースとしていた CbC-GCC を GCC-4.6 へのアップデートを行う。

2 Continuation basede C (CbC)

Continuation based C (以下 CbC) は当研究室で開発しているプログラミング言語である。CbC のプログラムはコードセグメント毎に記述され、コード間を軽量継続により処理を移るという特徴を持つ。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。

2.1 コードセグメント

CbC においてのプログラムの基本単位としてコードセグメントという概念がある。記述の仕方は C の構文と同じで、型に“ __code ”を使うことで宣言できる。コードセグメントへの移動は“ goto ”の後にコードセグメント名と引数を並べて記述することで行える。図**??**はコードセグメント間の処理の流れを表している。

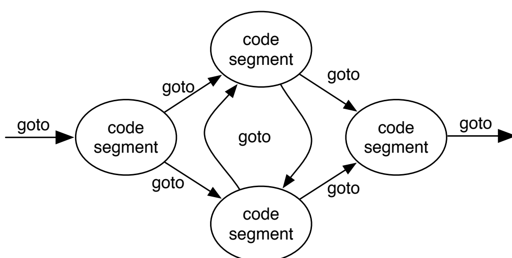

図 1: コードセグメント間の継続 (goto)

2.2 軽量継続 (light-weight continuation)

コードセグメントは C の関数と違って返り値を持たず、処理が終われば次のコードセグメントへと処理を移る。このコードセグメント間の継続制御を軽量継続 (light-weight continuation) と呼ぶ。C 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。だが、返り値を持たないコードセグメントではスタックに値を積んでいく必要な無く、最小限のスタックの使用ですむ。

軽量継続によりループ制御、関数コールとスタックの操作を意識し最適化がソースコードレベルで行えるようになる。

3 GCC-4.6 への実装

3.1 軽量継続の実装

CbC はコードセグメント間の処理の移りを軽量継続で行う。その実態は、アセンブラでの関数の呼び出しを call ではなく jmp で行うようにするというものである。

3.1.1 Tail Call Elimination

GCC には最適化の 1 つに Tail Call Elimination (末尾除去) がある。Tail Call Elimination は関数の呼び出しを call ではなく jmp で行い、返り値を大元の関数に返すというものである。図**??**は Tail Call Elimination によるプログラムの処理の流れを表す。

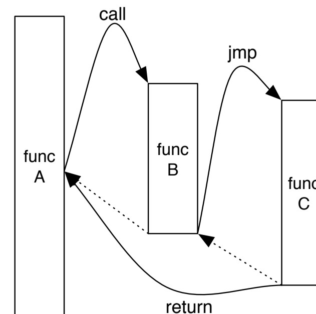

図 2: Tail Call Elimination の例

CbC における軽量継続の実装はこの Tail Call Elimination を用いて行われている。コードセグメントは全てこの Tail Call Elimination にかからなければならない。だが、CbC-GCC-4.5 において Tail Call Elimination にかからないコードセグメントがあることを確認できた。この点は GCC-4.6 へのアップデートに合わせ改善する。

3.2 typedefrec の実装

C では関数や構造体の宣言の時に自分自身を引数にすることができない。そこで“ typedefrec ”という構文を作り、図??のような宣言を行えるようにしたい。

```
typedefrec void *funcA(int, funcA);

typedefrec struct {
  node left;
  node right;
} *node;
```

図 3: typedefrec の例

より柔軟なプログラミングが行えるように typdefrec を実装を行いたい。

3.3 環境付き継続

CbC では通常の C の関数からコードセグメントに継続する際、元の C の関数に処理を戻すことがように環境付き継続を実装してある。環境付き継続は __return 変数を参照することで用いることができる。__return 変数は参照されると、参照した関数のアドレスを覚えておく。コードセグメントの継続の際に引数に __return 変数を渡すことで、関数の呼び出し元のアドレスも渡すことができる。後は引数として渡されたきたアドレスへ飛ぶことでいつでも C の関数に戻ることができる。

3.3.1 __return 変数の問題

しかし現在この __return の値は static で実装されている。これではスレッドセーフであるとは言えない。マルチスレッドで__return を扱うと、元の関数に戻る前に __return の値が書き換えられる可能性があるからである。そこで、__return をスレッドセーフにする必要がある。

3.4 x86_64 での fastcall

GCC では関数の呼び出しの際に引数はスタックに積まれて渡されるが、レジスタを使って渡すようにする fastcall という拡張機能がある。CbC-GCC ではコードセグメントとして宣言された場合 fastcall が自動で付くようにしていた。しかし、x86_64 においてこの fastcall は標準の機能となっており、コンパイルの際に warning が吐かれた。そこで、x86_64 の場合は fastcall を付与させないようにした。

4 CbC の今後

現在 CbC は C をベースとして設計されている。しかし、C ではプロトタイプ宣言や継続の際に型推論が扱えないなど不便な点があることがわかっている。そこで、Go や D 言語と言った言語へ実装を行いたいという要求がでてきた。

また、LLVM ベースの CbC コンパイラについても検討している。

5 現状と今後の課題

今後は本稿でも述べたとおり CbC コンパイラの実装を行なっていく。また、実装後は、32 ビット,64 ビットそれぞれでコンパイルしたプログラムの比較、それと Micro-C との性能比較も行う予定である。

Go 言語や D 言語への CbC の移植, LLVM ベースの CbC コンパイラについては実装の方法から考えていくことになる。

参考文献

[1] 河野真治: “ 継続を基本とした言語 CbC の gcc 上の実装 ”. 日本ソフトウェア科学会第 19 回大会論文集, Sep, 2002

[2] 河野真治: “ 継続を持つ C の回言語によるシステム記述 ”. 日本ソフトウェア科学会第 17 回大会論文集, Sep, 2000

[3] 与儀健人, 河野真治: “ Continuation based C コンパイラの GCC-4.2 による実装 ”. 琉球大学　情報工学科 学位論文, 2008

[4] 与儀健人, 河野真治: “ 組み込み向け言語 Continuation based C の GCC 上の実装 ”. 琉球大学大学院　理工学研究科 学位論文 (修士), 2010

[5] 下地篤樹, 河野真治: “ 線形時相論理を用いた Continuation based C プログラムの検証 ”. 琉球大学大学院　理工学研究科 情報工学専攻 学位論文 (修士), 2008

[6] GNU Compiler Collection (GCC) Internals: “ http://gcc.gnu.org/onlinedocs/gccint/ ”