C＋＋コンパイラのリンカ部における最適化技法

6Z－5

丹後田愛, 太田 裕, 林田 聖司

株式会社 東芝

1 はじめに

高級言語のコンパイラは、一般に構文解析を行ない中間コードを出力するパーサ部、中間コードよりアセンブラコードを作成するコード生成部、アセンブラそしてリンケージエディタより成り立っている。

本論文のターゲットであるリンケージエディタ(以下リンカとする)とは、コンパイラやアセンブラで分割して作成された複数のリロケータブルなオブジェクトファイルのリンクを行ない、メモリ配置を行なって一つの実行可能なオブジェクトファイルを作成したり、または、再配置可能なリロケータブルなオブジェクトファイルを作成するプログラムのことである。

また近頃では、PCやWSの分野だけでなく組み込みの分野でもC++を利用しようという動きがある。この分野での利用を考える場合には、コードをフルに最適化し、サイズを小さくする必要がある。そこで本論文では、コンパイラのリンカ部における冗長なコードを削除する最適化技法について述べる。

2 最適化の概要

C++はCの仕様を拡張したオブジェクト指向言語である。C++で追加されたいくつかの仕様により、プログラム中にはないコードをコンパイラの判断により生成することがある。コンパイラにより作成される関数には、暗黙定義関数(デフォルトコンストラクタ、デストラクタ、コピーコンストラクタ、代入演算子)、仮想関数のthis調整関数、インライン関数の実体があり、データでは、仮想関数テーブルがある。C＋＋のプログラムを分割してコンパイルした場合には、上に挙げたコードが、各翻訳単位毎に生成されることになる。

その結果、各翻訳単位毎に同じクラスのための暗黙定義関数や仮想関数テーブル、インライン関数の実体のコードが生成される可能性がある。

ここで、デバッグ情報を用いて、重複するコードの削除をリンカで行なう。この最適化により実行時間は向上しないが、冗長なコードをなくし、コードサイズの削減を行なうことが出来る。

3 コンパイラによるコード生成の事例

前節に、コンパイラによりコードが生成されることについてふれたが、この節では、仮想関数テーブルを除いた、3つの事例を以下に示す。

3.1 staticなインライン関数のコード生成

インライン関数は、通常は関数コードをコンパイル時に展開するので、関数の実体を生成する必要は無いが、プログラムの実行上必要になる場合がある。以下にその一例を示す。この例のように、アドレスが参照された場合は実体が必要である。

```
static inline int  f(){
    return 1;
    }
void main(){
   int(*p)() = f;
   }
```

3.2 仮想関数のthis調整関数

この関数は、仮想関数呼び出し時に、仮想関数へ渡すthisポインタのオフセットを計算する関数である。コンパイラがこの関数を生成するプログラム例を示す。

```
struct A{  virtual void f(); };
struct B{
    virtual void f();
    virtual void g();
    };
struct C:A,B{ void f(); };
void main(){
    C * pc = new C;
    B * pb = pc;
    pb->f(); // f() コール時の this 引数の
             // オフセットの計算が必要となる
    }
```

3.3 暗黙定義関数

暗黙定義関数とは、ユーザがプログラムの処理上必要な関数を作成していない時にコンパイラが暗黙に作成する関数のことである。暗黙定義関数には、デフォルトコンストラクタ、デストラクタ、代入演算子、コピーコンストラクタの4つがある。以下に暗黙定義関数が作成されるプログラム例を示す。

```
struct A{
    A(void);
    ~A(void);
    A(const A &);
    A & operator =(const A &);
    };
struct B : A{ };
void main(){
    B  b;              // デフォルトコンストラクタの作成
    B * pb  = new B; // デフォルトコンストラクタの作成
    B  p = b;          // コピーコンストラクタの作成
    p = b;             // 代入演算子の作成
    delete pb;         // デストラクタの作成
    }
```

4　最適化の方法

コードの削除をリンカで行なう際問題になるのは、どのようにして、同一のコードであるかを識別することである。本論文では、この識別するための情報として、デバッグ情報を用いることを提案する。コードの削除の際にデバッグ情報を用いることにより、型の検査を行なったりやコードの一致を調べたりせずにコードやデータの削除が行うことができる。

4.1　付加される情報

以下に、コンパイラにより生成されたコードやデータを識別するのに必要な情報を挙げる。これらの情報を元に、リンカでコードの削除を行なうことができる。また、ファイル情報やハッシュ値の必要性については 4.2、4.3 で述べる。

1. インライン関数
   (a) 関数の名前
   (b) 関数が定義されたファイル名
   (c) 関数の定義中のプリプロセッサに対応するためのハッシュ値
2. 暗黙定義関数と this 調整関数
   (a) 関数名
   (b) クラス名
   (c) クラスが定義されたファイル名
   (d) クラスの定義中のプリプロセッサに対応するためのハッシュ値
3. 仮想関数テーブル
   (a) クラス名
   (b) クラスが定義されたファイル名
   (c) ハッシュ値

4.2　ファイル名について

関数名やクラス名は、コードやデータを識別するための物であるが、その他にファイル名の情報を利用するのは、異なるファイルに書かれた同じ名前のクラスや関数を識別するためである。

例を挙げると、分割コンパイルの際、異なる名前のファイルをインクルードして、同じ名前で内容の異なるクラスを同一のコードとみなす危険性をファイル名の情報が防いでいる。しかしこのことは、同時に異なるファイルに書かれた、同じ名前で同じ内容のクラスや関数は最適化の対象とならないことを示している。

しかし、C++ では通常ヘッダファイルなどに class やインライン関数の定義を記述し、それをインクルードして使用する場合が多い。異なるファイルに書かれた同じ名前で同じ内容のコードを削除するためには、型の比較やコードの比較が必要となり、実装時の処理時間が増える。そのような少ない例のために処理時間が増えるのは効率が悪いので、通常に当てはまらない書き方をした場合には、削除の対象としないことにした。

4.3　ハッシュ値について

ハッシュ値を識別に利用するのは、前にも述べてあるようにプリプロッセサに対応するためである。クラスや関数の定義の中にプリプロセッサがあった場合には、定義時のライン情報が変わってくる。このハッシュ値は、ライン情報を元にして作られているので、プリプロセッサがある場合は、ハッシュ値を比較することによって識別ができる。

5　結論

C++ のコンパイラの最適化として、リンカで行なう最適化について述べた。リンカでコードの削除を行なうことにより、無駄なコードの無い、より小さなコードを生成することが出来た。また、コードの削除の際にデバッグ情報を用いることにより、型の検査を行なったりやコードの一致を調べたりせずにコードやデータの削除が行なえた。最適化の効果については発表で述べる。