Ｃ＋＋コンパイラの中間コード生成と最適化技法

2N－2

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、一般的に、前段部の中間コード生成部と、後段部のアセンブラコード生成部から構成される。中間コード生成部は、各種高級言語を構文解析し、言語に依存しない中間コードを生成する。また、アセンブラコード生成部は、中間コードを入力し、ターゲット機械依存のアセンブラコードを生成する。

本論文では、Ｃ＋＋コンパイラの中間コード生成部におけ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＋＋は非常に記述性が高く、Ｃ＋＋でのプログラミングによりソフトウエアの開発効率を向上することができる。このため、Ｃ＋＋は徐々に普及しつつある。

この反面、Ｃ＋＋で作成したプログラムの実行効率の低下が問題となる。まず、関数コールが頻繁に行なわれるため、オーバーヘッドが増大する。また、メモリアクセスの回数が増大したり、コードサイズが増大したりする。更に、このような現象がプログラマの意図しない場合にも起こり得るというのが非常に厄介な問題である。

例えば、一時オブジェクトの生成と消滅、メンバー関数の暗黙の定義、などは、プログラマが意図しない場合に起こり得る可能性が十分あり、多少の記述の相違によっても大きく実行効率が変わってくるため、実行効率を問題にする上では非常に重要である。

これらの事項については、中間コード生成部において、構文解析の段階で最適化が可能である。以下、中間コード生成部で可能な最適化技法について述べる。

なお、以下で使用する中間コードは原則として

演算子,src1,src2,result

の形をした3番地コードを使用する。

3　クラス型演算

3.1　ビットコピー演算

Ｃ＋＋で新たに「クラス型」が導入されたことにより、Ｃ＋＋プログラムではクラス型演算が非常に頻繁に出現する。例えば次のようなもっとも簡単な記述を考えてみる。

```
struct A { int a,b,c; }a,b,c;
a = b = c;
```

この場合、出力される中間コードは、

```
=,@b,@c,T1
=,@a,T1,T2
```

のようになる。ここに現れるT1,T2のうちT2は定義されるのみで使用されず、最終的に消去されるため、「一時オブジェクト」にはならない。一方T1は定義された後使用されるため「一時オブジェクト」である。

以下、クラス型の一時変数を「一時オブジェクト」と呼ぶことにする。一時オブジェクトは必要に応じて局所的に作成され、不要になった時点あるいはスコープが終了した時点で消去される。一時オブジェクトに関して問題になるのは、そのオブジェクトの型であるクラス型がコンストラクタ、デストラクタを持つ場合には、生成された一時オブジェクトの数だけこれらの関数が呼ばれることである。また、一時オブジェクトの領域として、アセンブラコード生成部で通常スタック領域が割り当てられるため、メモリ効率も悪化する。したがって、一時オブジェ

クトは、必要でない限りはできるだけ生成しないほうがよい。

通常の3番地コードでは、一時変数として演算の結果を使用できるように「一時的な結果」が設けられているが、ことクラス型に関しては不必要な「一時的な結果」は設けないのが賢明である。

ここで、ブロック転送の中間コード

MEMCPY,dst,src

を導入する。これは2番地コードといえる。これを用いることにより、上のコードは

```
MEMCPY,@b,@c
MEMCPY,@a,@b
```

といった中間コードに変換することができる。これによって、不必要な一時オブジェクトを消去できる。

3.2 三項演算子

不要な一時オブジェクトを作成しないわかりやすい例として、三項演算子を挙げることができる。通常の非クラス型演算では

```
int a,b,c,i;
a = i ? b : c;
```

のコードは、

```
!=,@i,#0,T1
FALSE,T1,,L1
=,T2,@b,T2
JMP,L2
LAB,L1
=,T2,@c,T2
LAB,L2
=,@a,T2,T3
```

のような中間コードに変換される。一時変数T2を使用することによる効率の低下はない。

クラス型演算の場合は、一時オブジェクトを用いずに、クラス型のポインタ型の一時変数を用いることにより効率化を図る。

```
struct A { int a,b,c; }a,b,c; int i;
a = i ? b : c;
```

のコードは、

```
!=,@i,#0,T1
FALSE,T1,,L1
ADDR,@b,,T2
=,T3,T2,T3
JMP,L2
LAB,L1
ADDR,@c,,T4
=,T3,T4,T3
LAB,L2
MEMCPY,@a,*T3
```

のような中間コードに変換される。

4 実装依存の最適化

浮動小数点などの型をもつオブジェクト割り当て方法は、一般的にはターゲット機械に依存しているため、出力すべき中間コードもターゲット機械に依存する。例として、以下、ターゲット機械が浮動小数点レジスタをもたない場合を考えてみる。

この場合、浮動小数点型のオブジェクト割り当てに関しては、クラス型と同様、集成型として扱う必要がある。従って、浮動小数点型の一時変数は「一時オブジェクト」に準じた扱いを必要とする。

例えば、前述の例のように単純代入を考えてみる。

double a,b,c;　　a = b = c;

これは次のような中間コードに変換される。

```
=,@b,@c,T1
=,@a,T1,T2
```

この場合もブロック転送の中間コードを用いて

```
MEMCPY,@b,@c
MEMCPY,@a,@b
```

とすることにより、不要な一時変数を消去できる。また、三項演算子についてもクラス型同様の最適化が可能である。

5 おわりに

以上に挙げた最適化は、最も簡単な技法である。講演では、一時オブジェクトに関する最適化、コンパイラが暗黙に定義する関数の実装方法、などについて述べる。