5P-5

コード最適化のためのCソースコンバータ

＊(株)東芝　青梅工場　　中村　宏明，黒髪　正人
＊＊株式会社　アーク情報システム　　飯塚　強

1．はじめに

セグメント方式をとるシステムにおいては、セグメントをまたがるデータの参照にかかる時間は一般的に速くない。そのようなシステムでC言語を使うとき、外部変数の参照に対する実行時間が問題となることがある。すなわち、C言語における外部変数はどのソースファイル上で定義してもよく、またどのソースファイル上からでも参照してもよいので、定義が複数のセグメントに存在し得るからである。このため、そのような外部変数参照に対する効率をあげる必要がある。

ここでは、我々が使用しているシステム上で、掲記の問題に対して試みた一つの方法について説明する。

2．既存の処理方法

一つのソースファイル内で定義された外部変数は、ローカルブロック（ソースファイル内で定義された関数が主に使用するスタティックデータブロック）内に、連続した領域として割り当てる。コンパイル時にはローカルブロック内の適当な位置を指す外部ディスクリプタ（リンク時にその位置に対するポインタ値が設定される領域）を生成し、関数の入口で、その外部ディスクリプタの内容をベースレジスタにロードする。外部変数の参照は、そのベースレジスタとオフセットにより行う。この場合、外部変数に対するロード命令は数回程度である。

しかし、他のソースファイル内で定義された外部変数に対しては、各外部変数が複数のソースファイルで定義されているので、一つのベースレジスタで複数の外部変数を参照するためのオフセット値をコンパイル時に求めることができない。従って、各外部変数ごとに外部ディスクリプタを生成しなければならない。そして、参照を行う直前に、その外部ディスクリプタの内容をベースレジスタにロードし、そのベースレジスタとオフセットにより参照を行う。すなわち、外部変数を参照する数だけロード命令を実行しなければならない。

3．外部変数参照の効率化

セグメントをまたがるデータの参照にかかる時間は、ベースレジスタにポインタをロードするときに費やされる時間が大半を占める。すなわち、参照を効率的に行うためにはロード命令を最少限に抑えればよい。そのためには、他のソースファイルで定義された外部変数を参照するときに効率のよい処理方法をとる必要がある。

そこで次のような方法をとることとした。

すべての外部変数をコモンブロック（複数のソースファイル内で定義された関数が共用するスタティックデータブロック）内に、連続した領域として割り当てる。コモンブロックは、ロードモジュールに対して複数定義することができ、外部変数を参照する各ソースファイルは必要なコモンブロックを参照することができる。このときコンパイラは、各ソースファイルにおいて参照するコモンブロックに対する外部ディスクリプタを生成し、リンカによりその値は設定される。実行時には、関数の入口で、その外部ディスクリプタの内容をベースレジスタにロードすればいいのでロード命令は数回で済む。

外部変数をコモンブロックに割り当てるためには、ソースファイルごとに定義されている外部結合を持つ外部変数定義を取り出し、一つにまとめ、そのまとめた外部変数を各ソースファイルで定義する必要がある。その処理を行うのが次に説明するCソースコンバータである。

4．Cソースコンバータ

4.1 処理概要

Cソースコンバータは、複数のCソースプログラムを読み込み、外部結合を持つ外部変数定義を取り出し、一つのヘッダファイルにまとめる。外部変数を定義していたソースプログラムからはその定義を削除し、作成したヘッダファイルを参照するように変換を行う。

図1に、大まかな処理の流れを示す。

C source converter for code optimization
＊ Hiroaki NAKAMURA, Masato KUROKAMI -- TOSHIBA Corp.
＊＊Tsuyoshi IIZUKA -- Ark Information Systems INC.

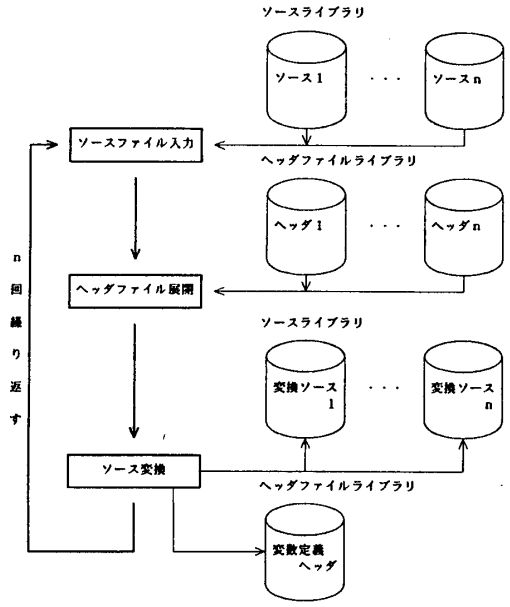

図1　処理の流れ

4.2 外部変数定義ヘッダファイル

作成するヘッダファイルの内容は、次のようになる。

ファイル名：cdata.h （変更可能）

```
asm("  .comm  cdata ");
  ソース１の外部変数定義
        ⋮
  ソースｎの外部変数定義
asm("  .ecomm  ");
```

.comm と.ecommは、この間で定義された変数をコモンブロックに割り当てるためのアセンブラ命令である。

一方、Ｃソースプログラムの外部変数定義を削除し、さらにソースプログラムの先頭に次の２行を付加する。

```
#include   "cdata.h"
#line      1     "ソースファイル名"
```

4.3 変換規則

ソースコンバータでは、次の規則に従いソースを変換する。

(1) 原則として、外部結合を持つ外部変数定義を取り込む。

(2) ファイルスコープのtypedef,struct（タグ名付きに限る),enum(タグ名付きに限る）を取り込む。

(3) 異なるファイル間に同名異型のtypedef,struct,enumがあった場合、後に宣言された方はtypedefやタグ名を変えて宣言する。

(4) 初期値として、外部変数のアドレスを与えている外部変数定義は取り込まない。

(5) 外部変数定義と内部結合を持つ外部変数又は関数名に衝突がある場合、警告を出す。

4.4 変換例

入力ソース１

```
1   int gdata1;
2   static   int  sdata1;
3   extern   int  edata1;
4
5   typedef  int  word;
6   word     tdata2;
7   extern struct st
8       {
9       char  i;
10      short j;
11      }     edata2;
12  struct st gdata2;
13
14  int  *list[] = {&sdata1,&edata1};
15  test1()
16      { /* ... */ }
```

入力ソース２

```
1   int edata1;
2   extern   int  gdata1;
3   struct   {
4            char  i;
5            short j;
6            }  edata2;
7
8   extern struct st
9       {
10      char  m;
11      short n;
12      }    gdata3;
13  test2()
14      { /* ... */ }
```

変換ソース１

```
1   # include "cdata.h"
2   # line 1 "S1/test1"
3
4   static   int  sdata1;
5   extern   int  edata1;
6
7
8
9   extern struct st
10
11
12
13      edata2;
14
15
16  int  *list[] = {&sdata1,&edata1};
17  test1()
18      { /* ... */ }
```

ヘッダファイル

```
1   asm("    .comm cdata");
2   # line 1 "S1/test1"
3   int gdata1;
4   # line 5 "S1/test1"
5   typedef  int  word;
6   # line 6 "S1/test1"
7   word     tdata2;
8
9   # line 7 "S1/test1"
10  struct st {
11      char  i;
12      short j;
13      };
14  # line 12 "S1/test1"
15  struct st gdata2;
16
17  # line 1 "S1/test2"
18  int edata1;
19  # line 3 "S1/test2"
20  struct   {
21           char  i;
22           short j;
23           }  edata2;
24
25  # line 8 "S1/test2"
26  struct stA {
27      char  m;
28      short n;
29      };
30  asm(" .ecomm");
```

変換ソース２

```
1   # include "cdata.h"
2   # line 1 "S1/test2"
3
4   extern   int  gdata1;
5
6
7
8
9
10  extern struct stA
11
12
13
14      gdata3;
15  test2()
16      { /* ... */ }
```

5．おわりに

このＣソースコンバータは、既存のソースファイルの内容を利用者が手を加えることなく、プログラムの実行効率を向上させる目的で作成した。当然のことながら、この変換は他のソースファイルで定義された外部変数を頻繁に参照しているプログラムほど効果を発揮する。

問題点としては、次の点があげられる。

・Ｃソースコンバータは複数のソースファイルから外部定義を取り込む必要がある。そのため、ある一つのソースファイルの外部変数定義を変更したとき、それに関わるすべてのソースファイルを変換しなおし、リコンパイルしなければならない。

・結果として二つのソースファイルができるので、保守が行いにくい。

今後は、このような最適化処理をコンパイラで実現する方式を検討していく予定である。