4F－6　ＰＳＳ：Ｃ言語プログラムの移植支援システム－継承支援Ｃコンパイラ

西川 一紀　川村 耕基　西風 一　中野 和俊

沖電気工業株式会社

1．はじめに

我々は移植支援システムＰＳＳ[1]の一部として移植阻害要因への対処方法を備えた継承支援Ｃコンパイラを開発した。
その具体的機能は、①エンディアン差異の吸収、②アラインメント差異の吸収、③ストラクチャ・パディングルール差異の吸収である。
本稿では継承支援Ｃコンパイラにおける移植支援方法及び、その評価について述べる。

2．移植阻害要因

移植阻害要因は、エンディアンの差異、アラインメントの差異、ストラクチャ・パディングルールの差異の３点が挙げられる。これらの移植阻害要因については[1]で述べられている。

ＰＳＳを用いて移植を行う場合、移植阻害要因は継承支援矛盾チェッカ[2]でを摘出し、継承支援エディタを用い移植阻害要因を隠ぺいするコーディングを行う。ＰＳＳにおいて継承支援Ｃコンパイラは、移植阻害要因への対処方法を備え、取り省くことが役割である。

3．継承支援Ｃコンパイラ

3．1　言語処理系による移植支援

言語処理系による移植支援において、言語仕様に要請されていることがらを以下に述べる。

① 移植テクニックの形式化
移植阻害要因別に対処方法を提供し、同じ移植阻害要因は、常に同じ方法で隠ぺいできるようにする。
このことは移植の難しさの軽減に寄与する。

② 移植コーディング量の軽減
移植阻害要因を隠ぺいするためのコーディング量を軽減し、移植作業時間の短縮に寄与する。

③ 移植後のソースコードの読解性の確保
移植後のソースコードの読解性を確保し、保守性の向上、デバッグ作業に必要とする時間の短縮に寄与する。

3．2　継承支援Ｃコンパイラの機能

継承支援Ｃコンパイラは、２章で概観した移植阻害要因への対処方法として、３．１で示した要請事項を満たすことが望まれる。
著者らは、数種類の案から識別子、または型定義名、または関数名の宣言に対するＣ言語の機能であるプラグマを用いたアーキテクチャ差異の隠ぺい指令を対処方法として選択した。

（１）　バイトオーダー差異への対処方法

① `#pragma big_endian({struct TAG, | union TAG, | enum TAG, | typename, | identifier,}*)`

{} *は０回以上の繰り返しを示す。

本指令は、指定された要素をビッグ・エンディアン形式で取り扱うようにコンパイラに指示する。
コンパイラは本指令を受け取ると指定された要素に対し、ビッグ・エンディアン形式でのロードまたはストアを実施するコードを生成する。

② `#pragma lillle_endian({struct TAG, | union TAG, | enum TAG, | typename, | identifier,}*)`

本指令は、指定された要素をリトル・エンディアン形式で取り扱うようにコンパイラに指示する。
コンパイラは本指令を受け取ると指定された要素に対し、リトル・エンディアン形式でのロードまたはストアを実施するコードを生成する。

（２）　アラインメント差異への対処方法

① `#pragma byte_access({struct TAG, | typename, | (*)*identifier,}*)`

{} *は１回以上の繰り返しを示す。

本指令は、指定された要素に対しバイト単位のアクセスを行うようにコンパイラに指示する。
コンパイラは本指令を受け取ると指定された要素に対し、バイト単位のロードまたはストアを実施するコードを生成する。
文字”＊”で示すポインタ修飾は、何度の参照を介した実体であるかを指示する。

② `#pragma half_access({struct TAG, | typename, | (*)*identifier,}*)`

本指令は、指定された要素に対し２バイト単位のアクセスを行うようにコンパイラに指示する。
コンパイラは本指令を受け取ると指定された要素に対し、２バイト単位のロードまたはストアを実施するコードを生成する。
文字”＊”で示すポインタ修飾は、何度の参照を介した実体であるかを指示する。
本指令は、オブジェクト効率に着目して提供した指令である。

PSS: The Porting Support System for C-programs - Porting Support C Compiler
Kazunori NISHIKAWA, Kouki KAWAMURA, Hajime NISHIKAZE and Kazutoshi NAKANO
Oki Electoric Industory Co., Ltd.

（3） ストラクチャ・パディングルール差異への対処方法

現版の継承支援Cコンパイラでは、この差異をコンパイル・スイッチで隠ぺいしているが、現在、以下に述べる対処方法を検討中である。

```
#pragma padding({struct TAG, |
                 typename, }*)
```

本指令は、指定された要素のパディングルールを移植元の機械のパディングルールと同等にするようコンパイラに指示する。

コンパイラは本指令を受け取ると指定された要素に対し、移植元の機械と同じパディングルールでメンバオフセットを計算し、必要ならばバイト単位のロードまたはストアのコード生成を実施する。

3．3 継承支援Cコンパイラの使用例

具体的な使用例を述べるために、以下の例を考える。

buffer ： 外部から入力されるビッグ・エンディアン形式のデータ。
data1 ：
data2 ：
data3 ： 外部から入力されたデータを格納する領域。（リトル・エンディアン形式。）

```
char * buffer;
int data1, data2, data3;

@1   data1 = *(int *)(buffer + 1);
@2   data2 = *(int *)(buffer + 2);
@3   data3 = *(int *)(buffer + 3);
```

図－1 移植前のソースコード

移植前のソースコード（図－1）おいて@1, @2, @3が移植阻害要因となる箇所である。

具体的な移植阻害要因を@1の箇所を例に取り説明する。

data1はリトル・エンディアン形式であり、bufferの指し先はビッグ・エンディアン形式であるためバイトオーダーの差異が生じる。

@1は代入文であるため、右辺に対し左辺の型に一致したロード命令が行われる。しかしロードの対象となるロケーションは4バイト境界である保証はなく、アラインメント例外が生じる可能性を含む。

本ソースコード（図－1）を従来一番簡単に行えると考える移植を行った場合と、継承支援Cコンパイラを用いて移植を行った場合の移植後のソースコードを以下に示す。

```
#define big_byte_ld(rval)  .....  ①
#define byte_st(lval, rval)  .....  ②

char * buffer;
int data1, data2, data3;

byte_st(&data1, big_byte_ld((int *)(buffer+1)));
byte_st(&data2, big_byte_ld((int *)(buffer+2)));
byte_st(&data3, big_byte_ld((int *)(buffer+3)));
```

図－2 従来の移植後のソースコード

①は、rvalをビッグ・エンディアン形式でバイト単位のロードを行う定義であり、②は、rvalをlvalで示す場所へバイト単位のストアを行う定義である。

```
#pragma big_endain(buffer)
#pragma byte_access(*buffer)

char * buffer;
int data1, data2, data3;

data1 = *(int *)(buffer + 1);
data2 = *(int *)(buffer + 2);
data3 = *(int *)(buffer + 3);
```

図－3 継承支援Cコンパイラによる移植後のソースコード

上記の移植後のソースコードは共に移植阻害要因を隠ぺいしているが、継承支援Cコンパイラを用いて移植した場合、移植コーディング量、移植後のソースコードの読解性に優位な点が認められる。

3．4 評価

継承支援Cコンパイラを用いて移植作業を行った場合の評価を述べる。

（1） コーディング量

従来の移植方法では図－2に見られるように移植阻害要因別に対処を行うものであった。

しかしながら継承支援Cコンパイラでは図－3に見られるように、同じ移植阻害要因はプラグマ指令を用い一度に対処できる。

このことは移植作業におけるコーディング量の軽減に寄与している。

（2） デバッグ作業に要する時間

デバッグ作業は、コーディング過程における不適切なコーディングによって生じる。

継承支援Cコンパイラによる移植方法はプラグマ指令の挿入であり、コーディング過程での作業はプラグマ指令の挿入がほとんどを占める。

このことからデバッグ作業は、挿入したプラグマ指令のシンタックスまたはセマンティクスエラーの復旧が主となる。

よってプラグマ指令に対する形式的なデバッグ手法を身につけることにより、従来よりデバッグ作業に要する時間は減少すると考える。

また、プラグマ指令を採用したことは図－3に見られるように、移植後のソースコードの読解性の確保に寄与していることは明白である。

4．まとめ

ＰＳＳの一部としての継承支援Cコンパイラの拡張文法仕様について概観した。

今後の方針として、より利用者のニーズに一致した移植支援用コンパイラの開発を行いたい。

参考文献

（1） 西風、原田、小林、前田 「ＰＳＳ：C言語プログラムの移植支援システム－概要」 本予稿集 4Ｆ－5
（2） 前田、大浜、西風 「ＰＳＳ：C言語プログラムの移植支援システム－矛盾指摘チェッカ」 本予稿集 4Ｆ－6