3D-3

Ａｄａサブセットコンパイラの試作

森本　真一*　菅野　高博**

*日本電気㈱ ﾏｲｸﾛｺﾝﾋﾟｭｰﾀｿﾌﾄｳｪｱ開発本部
**日本電気技術情報ｼｽﾃﾑ開発㈱

１．はじめに

本発表ではＵｎｉｘ上に実現したＡｄａのサブセットコンパイラについて述べる。　このコンパイラはＰＣＣ（Portable C Compiler ）と同じ形式の中間言語を持つ２パスコンパイラである。

Ａｄａには、ソフトウェアの信頼性や保守性を向上させる多くの機能が備わっている。そのため従来のＡｄａコンパイラはパス数が多い大規模なものが多くコンパイルに時間がかかるという欠点があった。これに対して、ここで述べるコンパイラは既存の（Ｃの）コンパイラと同様の構成にする事により、既存コンパイラと同程度のコンパイル性能を実現した。さらにＰＣＣのコード生成部を利用する事によりＰＣＣと同程度の実行効率を持つコンパイラを短期間で開発できた。

２．本コンパイラの機能と構成

現時点では、本コンパイラはＡｄａの機能の中で次の機能を除いた部分を実現している。

派生型、実数型、多重定義（の一部）、再名宣言、タスク、例外処理、分割コンパイル、汎用体、表現仕様

本コンパイラの構成は、図１の様になっている。

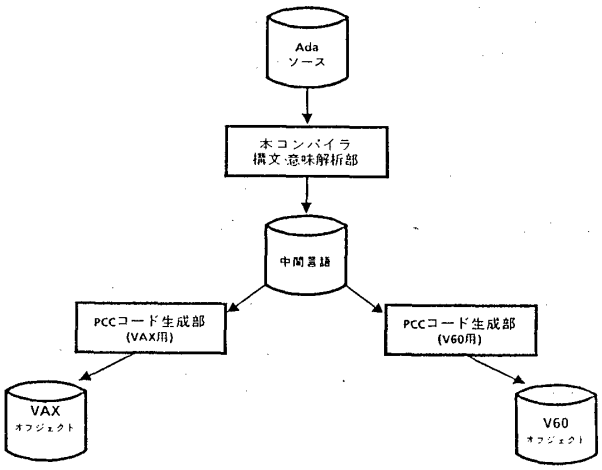

図１　本コンパイラの構成

マルチパスコンパイラでは、どの様なフェーズに分割するか、フェーズ間の中間言語をどの様な形式にするかが問題になる。本コンパイラはＰＣＣと同様のフェーズからなる２パスコンパイラである。２つのフェーズ間の中間言語はＰＣＣと同じ形式のものを採用している。

この構成には次の利点がある。

１）パス数を減らす事によりファイルとの入出力や中間言語の解析の回数が減りコンパイル時間を減少できる。

２）ＰＣＣと同形式の中間言語を用いる事によりＰＣＣのコード生成部を利用でき、開発量を減少できる。またＰＣＣのコード生成部は移植性や実行効率を考慮して開発されている［１］ので、新たに開発するより信頼性やコードの効率が良くなる。

３．中間言語の特徴

一般のマルチパスコンパイラの中間言語には、

ａ）ソース言語の構造を反映した木構造の言語

ｂ）オブジェクト(アセンブラ)の構造を反映した線形の言語

の２種類がある。　このコンパイラで用いる中間言語は次の様なハイブリッド形式（ａ）とｂ）の中間の形式）をしている。

- 制御構造はオブジェクト(アセンブラ)と同様に、アセンブラ命令とラベルで表わし、式（と代入文）は構文構造を木構造で表現する。

３．１　利点

この形式の中間言語を用いると次の利点がある。

１）式の部分だけが木構造で表現されるのでプログラム全体を木構造で表わす場合よりも必要となる記憶領域が少ない。

２）式は木構造で表現されるので多重定義は完全に解決できる（Ａｄａでは、多重定義は式単位で決定される）

３．２　問題点と対策

ＰＣＣ形式の中間言語を用いると、上で述べた利点があるがＣとＡｄａの言語仕様の違いによる問題点も存在する。ここでは問題点の内容と対策を述べる。

An Implementation of an Ada Subset Compiler

Shin-ichi Morimoto*　Takahiro Kanno**

* NEC Corporation, Microcomputer Software Development Lab.
**NEC Scientific Information System development Co. Ltd.

1）演算の不対応

Ａｄａには、配列の代入や所属判定演算（変数の値が指定された範囲内にあるかを判定する演算）の様にＣには直接対応するものがない演算がある。ここではＣの演算の組み合わせで実現できるもの（所属判定演算など）はその組み合わせによって表現し、実現できないもの（配列の代入など）は実行時ルーチンによって実現している。

2）型の表現方法

ＰＣＣ形式の中間言語では、各ノードの型を表わす要素はそのノードの型の構造を表わしている。しかしＡｄａでは型は構造ではなく名前によって判断されるので、ＰＣＣの形式をそのまま用いる事はできない。ここでは各ノードの型を表現する要素はそのノードの型の名前（シンボルテーブルの添字）を表わし、中間ファイルに出力する際にＰＣＣと同様に型の構造を表わす様に変更している。

3）プロローグ処理

Ａｄａでは副プログラムの内で副プログラムを宣言する事ができる。このため副プログラムの実行中には、それより（プログラムの）文面で外側にある副プログラム（やパッケージ）のframe pointer の値を知る必要がある。ここではstack frame 上に直接外側の副プログラムを指すリンク(static link) を持っていて呼出し時に呼出された副プログラムの外側の副プログラムに指すリンクをはる様にプロローグを変更している。

4）副プログラムのラベル

Ｃでは同じ名前の副プログラムは１つしか宣言できないので、ＰＣＣでは副プログラムの名前をそのまま中間言語上でその入口を示すラベルに用いている。しかし　Ａｄａでは同じ名前の副プログラムを複数個宣言できるので、その名前とコンパイラが生成した数値（これは一意に定まる）を組み合わせたものを入口を示すラベルとして用いている。

4．コンパイラの性能評価

ここでは本コンパイラの性能をＰＣＣと比較して評価する。　表１は本コンパイラとＰＣＣが、素数を求めるプログラム［2］のコンパイルと実行に要した時間を示したものである。なおコンパイルと実行は、μVAX-2のULTRIX上で行ない、timeコマンドで測定した(usrの時間とsysの時間の合計を表示している)。

表１　コンパイラの性能比較

	ｺﾝﾊﾟｲﾙ時間(秒)	実行時間(秒)
本ｺﾝﾊﾟｲﾗ*1	３.２（３.１）	１２３（２９.８）
ＰＣＣ *2	２.７（２.６）	１６.１(２５.８)

＊１　()は実行時チェックなし
＊２　()はレジスタ変数不使用

コンパイル時間は、本コンパイラはＰＣＣより約２割増加しているがＡｄａの構文や意味の複雑さを考えるとほぼ同等といえる。　これはパスを少なくし、中間言語の形式を簡単にした事が原因と思われる。

実行時間は、本コンパイラはＰＣＣの約８倍であるが、次の点からそのまま比較する事はできない。

- Ａｄａでは配列の添字の範囲等の実行時チェックを行なっているが、Ｃでは行なっていない。
- Ｃのプログラムでは、レジスタ変数を用いる事により実行効率を向上させている。

そこで条件を対等にするため、Ａｄａでは実行時チェックを行なわず、Ｃではレジスタ変数を自動変数に書き換えて測定したところ、本コンパイラの実行時間はＰＣＣよりも約１割増加しただけであった。そこで次の改良を行なう事により、Ｃと同程度の実行効率を持つＡｄａコンパイラが実現可能と思われる。

1）実行時チェックの効率化

コード生成部に実行時チェックのための生成コードの記述を追加する（現在は、実行時チェックはすべて実行時ルーチンで行なっている）。

2）レジスタプラグマの実現

Ｃのレジスタ変数と同じ機能を持つプラグマ（コンパイラへの指示）を実現する。

5．まとめ

ＰＣＣと同程度のコンパイル効率および移植性を持つＡｄａサブセットコンパイラを短期間（１６人月）で開発できた。

今後は、４．で述べた改善やタスク等のＡｄａ特有の機能の実現を行ない、ＵＮＩＸ上でのＡｄａの実用性の評価を行なっていきたい。

6．参考文献

[1]木村　他：V60 ｱｰｷﾃｸﾁｬを生かした言語処理系
第３３回情報処理学会全国大会

[2]16ビット機用コンパイラ・ベンチマーク
日経バイト　11月号　1985　93～132