

自動ベクトル化並列化コンパイラ V-Pascal Ver.3

上原 哲太郎 國枝 義敏 津田孝夫

京都大学工学部情報工学教室
〒 606-01 京都市左京区吉田本町

あらまし

我々が開発中の自動ベクトル化並列化 Pascal コンパイラ「V-Pascal Ver.3」について述べる。V-Pascal Ver.3 は、ポインタ変数間のエイリアス解析、整数変数の値域の静的推定、再帰の手続きや while 型ループのベクトル化/並列化、多重ループのベクトル間接参照を用いたベクトル化/並列化など、多くの特徴的な解析機能、ベクトル化並列化機能をもつ。

和文キーワード 自動並列化、自動ベクトル化、値域推定、再帰の手続き、while ループ、多重ループ、ベクトル間接参照

An Automatic Vectorizing and Parallelizing Compiler V-Pascal Ver.3

Tetsutaro UEHARA Yoshitoshi KUNIEDA Takao TSUDA

Department of Information Science, Faculty of Engineering, Kyoto University.
Yoshida-honmachi, Sakyo-ku, Kyoto-city, Kyoto 606-01, Japan.

Abstract

We describe the design and implementation of our automatic vectorizing and parallelizing Pascal compiler "V-Pascal Ver.3" briefly. V-Pascal Ver.3 is equipped with many specific analyzing and vectorizing/parallelizing facilities such as precise analysis of aliases between pointer references, static estimation of value ranges of integer variables for dependence analysis, vectorization and parallelization of recursive procedures and while loops and vectorization and parallelization of multiply nested loops using indirect vector addressing.

英文 key words automatic vectorization, automatic parallelization, value-range estimation, recursive procedure, while loop,
multiply nested loop, indirect vector access.

1 研究の背景

Fortran90の登場 [1] により、これまで Algol 系言語の特徴であった while/repeat-until 型ループや再帰の手続きなどの制御構造、およびポインタを用いたデータ構造などが、大規模数値処理の分野でも利用可能になった。これらの制御構造やデータ構造を含むプログラムは記述性と可読性の面で優れているため、こういったプログラムの自動ベクトル化/並列化手法の研究は今後重要になると考えられる。

また、近年台頭してきた超並列型計算機環境においては、従来の逐次型言語で書かれたプログラムをそのまま並列化できるコンパイラは存在せず、並列用に拡張された記述を持つ逐次型言語か専用の並列型言語のコンパイラが提供される場合が多い。しかしこれらの言語を利用したプログラミングは未だに一般的ではなく、逐次型言語を自動もしくは半自動的に並列化するコンパイラへの要求は大きい。

そこで我々は、これらの要求に応えることのできる自動ベクトル化/並列化手法の研究を行なうプラットフォームとして、V-Pascal Ver.3 と名付けたコンパイラを開発している。本稿では、本コンパイラの概要と、このコンパイラを利用して研究されている自動ベクトル化/並列化手法について述べる。

2 V-Pascal Ver.3 コンパイラの概要

2.1 全体構成

V-Pascal Ver.3 の全体構成を図 1 に示す。

図に示すように、V-Pascal Ver.3 は、全体に 3 つのフェーズ (Phase) から成り立っている。Phase 1,3 はそれぞれ言語依存/機種依存の処理

を行ない、Phase 2 が汎用のベクトル化/並列化処理部となっている。

Phase 1 は、ソースプログラムを中間コード (Intermediate code) に変換する。現在 Pascal 用の Phase 1 が稼働中である。この Pascal は言語拡張を含まない、通常の逐次言語の仕様に基づいたものであるが、コメントによるユーザー指示が行なえるようになっており、現在は分散メモリ型並列計算機用のオブジェクトコード生成時に、データ割付けを指示するために用いている。

Phase 2 は、Phase 1 から受けとった中間コード中のベクトル化/並列化可能部分を検出し、変換する。この Phase 2 の内部は、さまざまなベクトル化/並列化手法を実現するモジュールの集合体である。これらの各モジュールは独立しており、追加、削除、適用順序変更などによってコンパイラの構成変更が可能になっている。また他に、これらのベクトル化/並列化モジュールを支援する各種依存解析モジュールなども、このフェーズに含まれる。

Phase 3 は、Phase 2 で並列化/ベクトル化された中間コードから各ターゲットマシン用のオブジェクトコードを生成する。内部では、機種依存の最適化処理も行なう。現在、日立製作所のベクトルスーパーコンピュータ S-3800 のアセンブリ言語によるプログラムを生成するものと、富士通研究所の高並列計算機 AP-1000 用の C 言語プログラムを出力するものの 2 種が稼働している。

これらの各フェーズの間のデータの交換に使われる中間コード (Intermediate code) は、無限個のレジスタと無限個/無限長のベクトルレジスタを持つ仮想的なベクトル計算機の命令コードと、それらを複数台並列動作させるための通信/同期用のプリミティブからなる。

本バージョンのコンパイラの開発にあたっては、UNIX OS 環境上で C++ を用いた環境を利用している。また、Phase 1 の作成や中間コード関連の処理部分には lex, yacc などのコンパイラコンパイラを利用している。コンパイラ各部で使われるデータを C++ のクラスを用いて記述することにより、記述の抽象性を大きく高め、開発効率を向上させることが可能になると考え

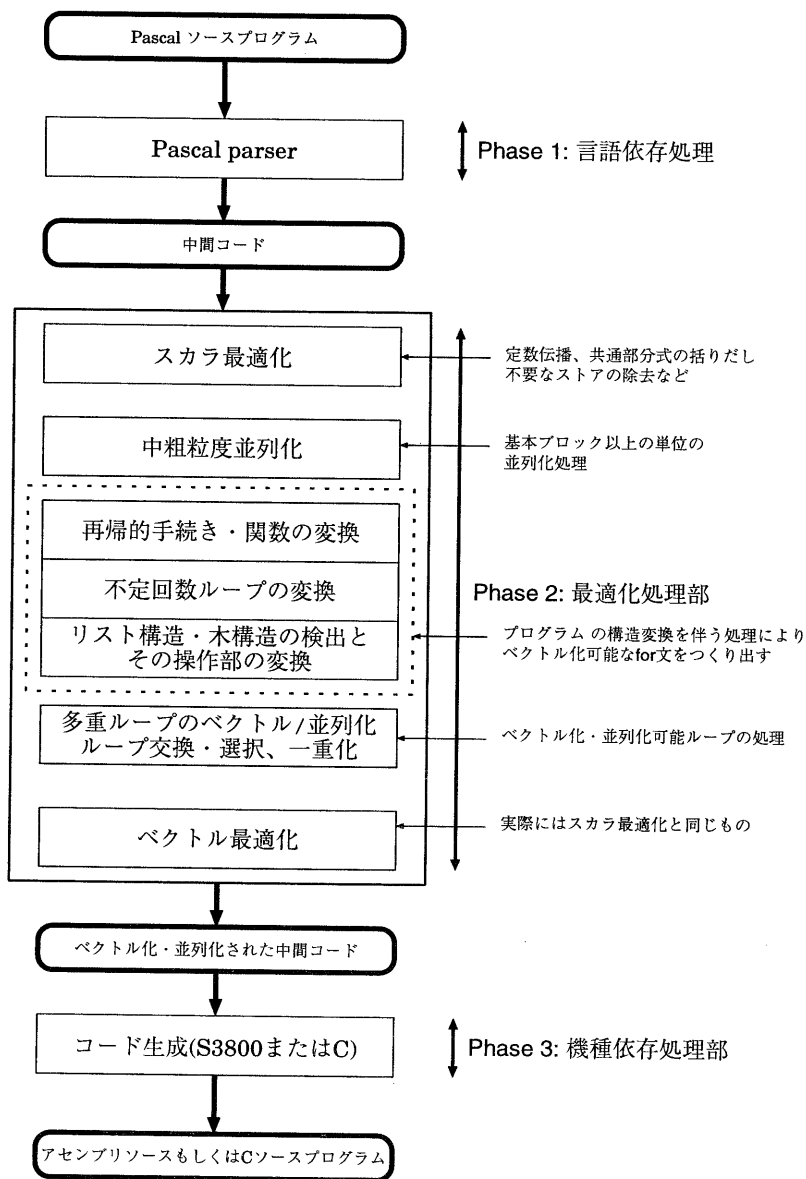


図 1: V-Pascal Ver.3 の全体構成

```

procedure F(var a, b : integer);
begin
  ...
end;
begin
  ...
  F(c, c); { (a, b) がエイリアスペアになる }
  ...
end;

```

図 2: 参照呼び引数によるエイリアスの例

```

var p, q :  $\uparrow$ integer;
begin
  ...
  new(p);
  q := p; { (p $\uparrow$ , q $\uparrow$ ) がエイリアスペアになる }
  ...
end;

```

図 3: ポインタ変数によるエイリアスの例

られる。

2.2 解析モジュール群

本節では、V-Pascal Ver.3 の Phase 2 の部分でのベクトル化/並列化モジュールに必要な、各種解析モジュールについて説明する。

2.2.1 エイリアス解析

異なる 2 つの変数がメモリの同一領域を指すとき、2 つの変数がエイリアス (alias) であるという。変数間にエイリアスが起り得る状況下では、ある変数への代入が他の変数の内容を同時に変更してしまう可能性があるので、プログラムのデータフロー解析の前には、あらかじめエイリアスを起こし得る変数の組 (これをエイリアスペア (alias pair) と呼ぶ) を確定しておかなければならない。このエイリアスペアが存在するか、存在する場合はどこかを調べる解析をエイリアス解析という。

言語 Pascal におけるエイリアスは、参照呼び引数 (Pascal では var 引数) を複数持つ手続きにおいてその複数の仮引数に同一の実引数が割り当てられた時 (図 2)、および 2 つ以上のポインタ変数が同一のメモリ領域を指す時 (図 3) などに起き得る。参照呼び引数の引き起こすエイリアスの解析に関してはこれまで多くの研究

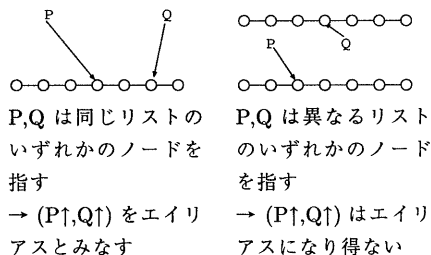


図 4: ポインタ変数によるエイリアス解析の例

があり、既に厳密な解析結果が得られるアルゴリズムが Cooper[2] によって提案されているので、V-Pascal Ver.3 ではそれを実現している。

ポインタ変数によって起り得るエイリアスの解析に関しては、これまでは困難とされており、多くのコンパイラの実装では全てのポインタ変数の指す内容の任意の組合せのうち、同じ型に属するものは全てエイリアスペアになり得ると判定するのが普通である。それに対し我々のグループでは、より精密で実用的なエイリアス解析の手法を提案済みである [3]。この手法では、ポインタ変数によって構成されるリストや木などのデータ構造の操作を行なう部分をプログラム中から検出、解析する。その結果、例えば図 4 の右側のように、異なるリスト構造のいずれかのノードを指すことがある P, Q というポインタ変数の組の内容はエイリアスになり得ないことが検出できる。

ただし、我々が発表済みのアルゴリズムでは、手続き間に渡る解析についての考察が厳密ではない。しかし V-Pascal Ver.3 では、手続き間に渡る各中間コードの制御依存関係を Intergrated Control-Flow Graph(ICFG) と呼ぶグラフで表現し、この ICFG 上で適用できるように改良したアルゴリズムを利用して、より厳密なエイリアス解析が可能になるようにしている。

2.2.2 制御/データフロー解析

制御/データフロー解析は、前述のエイリアス解析の結果を受けてより厳密に行なわれる。その結果は、1) 基本ブロック中の中間コード単位、2) ループ内の各中間コード単位、3) 基本ブ

ロック単位、4) 手続き単位の4つの異なる単位でそれぞれ表現し、ベクトル実行のような細粒度から手続き単位の並列実行のような粗粒度までの、異なる粒度の並列化に利用する。依存関係のグラフの表現には、D 行列と呼ぶ行列表現を用いている。

2.2.3 配列添字解析

ループ内で参照される配列の添字の解析は、ベクトル化や並列化に必須となる解析である。この解析は、変数値の範囲に関して条件のついた Diophantous 方程式の整数解の存在判定問題に帰着できる [4]。この解析手法に関しては、これまで多くの研究があり、配列添字式がループの制御変数と定数からなる線形式の場合には、厳密な解析結果が得られる手法がいくつも提案されている。我々も、この Diophantous 方程式の判定問題を整数解の範囲で厳密に解く手法を独自に提案済みである [5]。

しかし配列添字式に、ループ内では値が変わらない変数（これを記号定数と呼ぶ）が含まれる場合や、配列添字式が非線形式を含む場合には、静的で厳密な解析は難しいとされており、実行時にそれらの変数や式の値によって、可能な場合にのみベクトル/並列実行を行なう動的判定手法がとられることが多い。

これに対し V-Pascal Ver.3 では、既に提案済みの手法により静的解析および動的判定を行なう他に、後述の変数の値域推定によってあらかじめ記号定数の値の範囲を推定しておくことにより、記号定数が存在する場合でもより厳密に静的解析を行なう手法を実現する。

2.2.4 変数の値域推定

配列の添字解析において問題となるのは、記号定数が添字式に現れる場合である。例えば図 5 のように、変数 S の値が実行時までわからない場合、 S の値によって可能になる 2 種類のベクトル実行コードそれぞれを用意しておき、実行時に S の値によって切替えるような目的コードを生成することになる。

しかし、静的な解析によってたとえば S が常に 10 以上であることがわかれば、生成する目

```
var a : array [1..100] of integer;
for i:=1 to 10 do begin
  a[i+S] := ...
  a[i+10] := ...
end;
```

(a) 与えられたプログラム

$S \geq 10$	そのままベクトル実行可能
$S < 10$	2 参照入れ換えてベクトル実行可能

(b) S の範囲によるベクトル実行の場合分け

図 5: 記号定数を含む配列添字式の例

的コードは 1 種類で済む。このように、ある変数について、ループに到達した時の値の範囲が推定できれば、そのループの配列添字の依存解析に大きく役立つ。V-Pascal Ver.3 では、このような変数について値域の推定を試み、配列添字の解析能力の向上を図る手法を提案している [6]。

この値域情報を求めるアルゴリズムは、データフロー解析と似たものを利用している。その概略は以下の通りである。

1. 通常のフロー解析

中間コードを基本ブロック単位に分割し、コントロールフロー解析およびデータフロー解析を行う。

2. 添字参照情報の処理

中間コードの各ノードに対し、以下の処理を行う。

- (a) 中間コードに配列添字式が存在するかどうか探す。なければ以下は実行しない。
- (b) 添字式から記号定数の上限値と下限値、すなわち値域情報を計算する。この場合の値域情報は、ループ制御変数の上下限値と、配列添字の宣言から求める。
- (c) コントロールフローを逆にたどって添字式に現れる変数の定義点を探す。
- (d) その定義点に値域情報を加える。定義点に既に値域情報があれば、その値域情報との論理積を新たな値域情報としてその定義点に加える。

```

function    F(arg:arg-ty):retval-ty;
begin
  Shead; { 子環境への引数の計算部分 }
  if P1 then retval1 := F(arg1);
  if P2 then retval2 := F(arg2);
  if P3 then retval3 := F(arg3);
  :
  if Pn then retvaln := F(argn);
  Stail { 親環境への返り値の計算部分 }
end;

```

図 6: 正規化された再帰の手続き

3. 順方向フロー処理各ノードに存在する値域情報として IN, OUT 集合という 2 つの集合を定義する。

- (a) 中間コードのうち先頭にあるノードの IN 集合を ϕ とする。
- (b) 中間コードを先頭から順に制御の流れの方向にたどりながら、各ノードに対して、IN 集合と中間コードの文から新たな値域情報を得て、これを OUT 集合とする。OUT 集合から次のノードの IN 集合を求める。
- (c) 上の計算を、いずれのノードの IN 集合も変化しなくなるまで繰り返す。

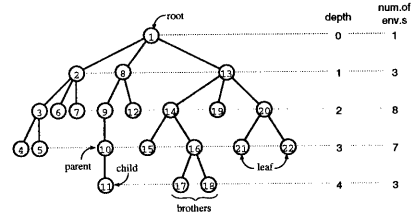
現在の実装では、この解析の対象となるのは、配列添字の記号定数となっている整数の単純変数であり、また値域情報としては単純な上限下限でなく、範囲推定の式を使う。また、順方向フロー処理は停止しない場合があるので、これはループ検出の上別途処理する。

2.3 ベクトル化/並列化モジュール群

本節では、V-Pascal Ver.3 の Phase 2 の部分でのベクトル化/並列化モジュール群について解説する。

2.3.1 再帰の手続きのベクトル化/並列化

再帰の手続きは、アルゴリズム記述を平易にする上で非常に強力な道具であり、多くのアルゴリズムの記述がこれによってなされている。V-Pascal Ver.3 では、我々が既に提案した「幅優先法」[7][8]に従って再帰の手続きの並列実



※ノードの番号は呼び出される順番を表す

図 7: 環境木

行可能部分を抽出し、変換する。その概略は以下のとおりである。図 6 のような形に正規化可能な再帰の手続きにおいて、再帰呼び出し毎に生成される局所変数領域を「環境」と呼ぶ。その環境毎の呼びだし関係は図 7 のような「環境木」で表されるが、この木において同じ深さにある環境間に互いに依存関係がなければこれらは並列に実行可能である。このことを用いて、環境間の依存関係を調べ、可能であればこれらの環境をベクトル実行もしくは並列実行する。

2.3.2 不定回数ループのベクトル化/並列化

不定回数ループとは、Pascal では while-do, repeat-until などにあたるものである。これらのループは、プログラムの実行がループの先頭に到達した時点でも、最終的な繰り返し回数が不明であるため、そのままではベクトル化/並列化できない。

このようなループのベクトル化手法については既に Wolfe[9] が提案した手法があるが、我々はこれを拡張して、より多くの不定回数型ループにおいて高いベクトル性能をひき出す手法を提案済みであり [10]、これを実装した。

この手法では、依存解析により、不定回数ループを、繰り返し回数を決定する部分とそうでない部分の 2 つのループに分割する (ループ分割)。後者のループは for 型のループ (既定回数ループと呼ぶ) となるので、通常的手法でベクトルまたは並列実行を試みる。また前者のループについても、ある定数回の繰り返し回数を仮定してベクトルもしくは並列実行した後、ループ終了条件が繰り返し途中で成立したか調べ、成立

していなければさらに回数を仮定して実行を継続する。終了条件が成立していたならば、余分な実行結果を除いたものを最終的な実行結果とする(ストリップマイニング)。

さらに、不定回数ループと既定回数ループで構成される多重ループについても、適切なループ交換により、より高速に実行出来るように変換する手法を提案しており [11]、これを実装中である。

2.3.3 多重ループの一重化によるベクトル化/並列化

多くの商用自動ベクトル化並列化コンパイラでは、多重の既定回数ループは最内側ループのみベクトル化し、外側のループは並列実行するのが通常であった。しかし我々は従来から、このような多重ループを、一重ループに変換して(これをループの一重化と呼ぶ)実行する手法を提案してきた [12]。一般にベクトルプロセッサはベクトル長が長いほど高い性能を出すことから、この一重化操作により性能の向上が期待できる。ただしこの一重化の際にベクトル間接参照の利用が必要である場合があるが、V-Pascal Ver.3 が主なターゲットとしている S-3800 の場合には、ベクトル間接参照の利用は性能低下は引き起こさず、この手法は有効である [13]。また制御変数の上下限式や配列添字式の中に記号定数を含む場合にも、ベクトル間接参照用のリストベクトルを静的に生成しておく手法を既に提案し [14]、実装中である。

2.4 分散並列アーキテクチャへの対応

V-Pascal Ver.3 は、ベクトルプロセッサが数台密結合したアーキテクチャを指向して作られている。しかし逐次プログラムのベクトル化手法と、超並列計算機の並列化手法には多くの共通項があり、転用が可能である。そこで V-Pascal Ver.3 の主要なモジュールを転用して、疎結合高並列計算機 AP-1000 向けのコンパイラの作成を行なった [15]。これを V-Pascal/DM と呼んでいる。V-Pascal/DM は、コメント行に記述されたコンパイラ指示に従って大規模配列を各要素プロセッサに分割配置した後、自動的に

ループの並列化、同期/通信文の挿入を行なう機能を持つ。

3 おわりに

V-Pascal Ver.3 は、我々がこれまでに提案してきた、自動ベクトル化、並列化コンパイラにおけるさまざまな解析、ベクトル化および並列化技法の効果を確かめるためのプラットホームとして開発されている。この目的のため、各解析部、ベクトル化/並列化部をモジュール化し、コンパイラの構成変更を容易にしている。本稿では現在実装完了もしくは実装作業中の各モジュールの機能について概略を述べた。

現在、V-Pascal Ver.3 は Phase 1 および Phase 3 についてはコーディングがほぼ完了しデバッグの段階にはいっているが、Phase 2 についてはまだ未完成的な部分が多く、現在鋭意作業中である。

謝辞

本コンパイラの開発にあたっては、多くの方の協力を得ています。設計にあたっては京都大学大型計算機センターの岡部寿男氏に多くの助言を頂きました。また本学在学中に開発作業に加わって下さった、また現在も協力下さっている方々の名前を以下に挙げてお礼にかえさせて頂きます(敬称略)。

梅田憲(現三菱電機勤務)、酒井淳嗣(現日本電気勤務)、藤田健治、三吉郁夫、久保田歩、韓 東洙、影本英樹、村井均、山本政行、上田康裕、野本政和、和田洋征

また、研究に際して議論下さった津田研究室の諸氏に感謝いたします。

参考文献

- [1] *Information technology - Programming language Fortran 2nd.ed.*, ISO/IEC 1259, 1991.

- [2] Cooper, K.D. *Analyzing Aliases of Reference Formal Parameters*, Conf. Rec. Twelfth ACM Symposium on Principles of Programming Languages(Jan.), pp. 281-290. 1985.
- [3] Matsumoto, A. and Tsuda, T. *Dependence Analysis between Pointer References in Pascal*, Proc. of International Symposium on Supercomputing (Fukuoka, Japan), pp.28-37, Nov.1991.
- [4] Zima, H.: *Supercompilers for Parallel and Vector Computers*, MIT Press, 1990.
- [5] 水沼, 上原, 岡部, 國枝, 津田: 記号定数および非線形式を含む多重ループのデータ依存解析, 日本ソフトウェア科学会第9回大会論文集, pp.485-488, 1992.
- [6] 酒井, 津田: 精密な依存解析のための変数値域の静的予測, 情報処理学会第48回(平成6年前期)全国大会 6G-4, 1994.
- [7] 上原, 津田: 「幅優先法」による再帰の手続きの自動ベクトル化・並列化, 並列処理シンポジウム JSPP'93, pp.135-142, 1993.
- [8] 岡山, 上原, 津田: 再帰の手続きの自動ベクトル化における幅優先法の拡張, 情報処理学会第48回(平成6年前期)全国大会 6G-3, 1994.
- [9] Wolfe, M. *Supercompilers for Supercomputers*, Addison-Wesley, 1990.
- [10] 末廣, 津田: WHILE 型ループの自動ベクトル化, 並列化, 情報処理学会第45回(平成4年後期)全国大会 分冊5, pp.51-51, 1992.
- [11] 村井, 末廣, 岡部, 國枝, 津田: ループ交換による while 型ループの自動ベクトル化, 日本ソフトウェア科学会第11回大会論文集, pp.65-68, 1994.
- [12] Tsuda, T. and Kunieda, Y., *Mechanical Vectorization of Multiply Nested Do Loops by Vector Indirect Addressing*, Proc. of the IFIP 10th World Computer Congress, Elsevier Science Publishers B.V.(North-Holland), pp.785-790, 1986.
- [13] Uehara, T. and Tsuda, T., *Benchmarking Vector Indirect Load/Store Instructions*, Workshop on Benchmarking and Performance Evaluation in High Performance Computing (Tokyo, Japan), pp.16-25, 1993.
- [14] 影本, 岡部, 國枝, 津田: 記号定数を含む多重ループの一重ベクトル化, 日本ソフトウェア科学会第11回大会論文集, pp.69-72, 1994.
- [15] 梅田, 上原, 津田: 分散メモリ型並列計算機用自動並列化コンパイラ V-Pascal/DM, 情報処理学会第48回(平成6年前期)全国大会 5G-1, 1994.