データフロー解析による関数型言語 Valid のコンパイル法†

——Datarol プログラムの抽出アルゴリズム——

立花 徹†† 谷口 倫一郎†† 雨宮 真人††

関数型言語を直接ハードウェアで実行するデータフロー・アーキテクチャに対して，通信やフロー制御のオーバヘッド，メモリレス概念の弱点といった問題が指摘されている．そのような問題を改善するアーキテクチャとして，著者らはメモリの概念を取り入れたマルチスレッド・コントロールフロー (Datarol プログラムと呼ぶ)に沿って並列演算実行する Datarol マシンを提案している．本論文では関数型プログラム Valid から，データ依存解析によって Datarol プログラムを抽出するアルゴリズムを述べる．次にそのアルゴリズムによって抽出されたいくつかの Datarol プログラムをデータフロー・プログラムと比較し，演算ノード数，ペアオペランドマッチング数等の点で優れていることを示す．またこれらの例題プログラムについて，関数型プログラム (LISP) や手続き型プログラム (C) を逐次型マシン向きにコンパイルしたコードと比較し，Datarol プログラムが有効であることを示す．

1. はじめに

動的なデータフローマシンではプロセススイッチがハードウェアで直接行われるので多重並行処理機構を効果的に実現できる．しかし，データフロー・アーキテクチャをそのままハードウェアで実現しようとすると，通信のオーバヘッド，フロー制御のオーバヘッド，メモリレス概念の弱点，といった問題が生じる．著者らは，データフロー・アーキテクチャのこのような欠点を改善するアーキテクチャとして，Datarol アーキテクチャを提案した[1),2)]．Datarol マシンは Datarol と呼ぶ多条演算系列に沿って並列実行し，超多重並行処理を効率的に行うことを特徴とするマルチプロセッサシステムである．Datorol マシンの機械語である Datarol プログラムはデータフロー・プログラムからペアオペランドの存在チェックやゲート演算などの余分なデータフロー制御を取り除いて最適化したマルチスレッドのコントロールフロー・プログラムである．

本論文ではデータフローマシン用に開発された関数型言語 Valid[3),4)] から，データ依存解析によって Datarol プログラムを抽出するアルゴリズムについて述べ，このようなデータ依存解析によって抽出した Datarol プログラムの評価を行う．評価としては Datarol プログラムとデータフロー・プログラム[5)] ならびに手続き的プログラムのコンパイルコードのステップ数，およびそのコード実行中に使用するレジスタ数を比較する．いくつかのプログラムについて比較した結果を示し，ステップ数，必要レジスタ数に関して Datarol プログラムが有効であることを示す．特にこれらの Datarol プログラムが，同一処理を行う手続き型プログラムのコンパイルコードと比べて遜色ないことを示す．

まず，第2，第3章では準備として Datarol マシンのアーキテクチャと Datarol プログラムの実行メカニズムを説明する．第4章では関数型プログラミング言語 Valid からデータ依存解析を基に Datarol プログラムを抽出するアルゴリズムについて述べる．このアルゴリズムは，Valid と同様の制御構造（ループ，条件分岐，関数呼び出し）を持つ ID[6)], SISAL[7)] といった関数型言語にも適用できると考えられる．ここでは特に，各命令の継続命令を決定するアルゴリズム，および各命令の変数に割り付けるレジスタを決定するアルゴリズムについて述べる．第5章では Datarol プログラムがデータフロー・プログラムに比べてどの程度の改善がなされているかをいくつかのプログラム例について調べる．ここで比較の対象とするデータフローマシンは NTT で開発された DFM[5)] を想定する．また，同じ例題プログラムについて，Lisp，ならびにC言語で記述したものをコンパイルしたコードの命令数と Datarol プログラムの命令数との比較を行う．

2. Datarol マシンのアーキテクチャ

2.1 システムの概要

図1に Datarol マシンの概略を示す．マシンは多数

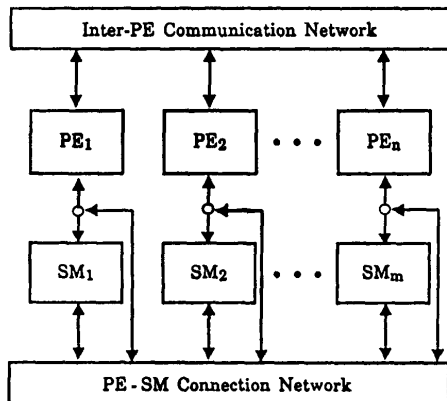
† Compiling Method of Functional Programming Language "Valid" by Data Flow Analysis—Extraction of Datarol Program—by TOHRU TACHIBANA, RIN-ICHIROU TANIGUCHI and MAKOTO AMAMIYA (Department of Information Systems, Interdisciplinary Graduate School of Engineering Sciences, Kyushu University).

†† 九州大学大学院総合理工学研究科情報システム学専攻

表1 命令セット
Table 1 Instruction set.

命令形式	実行内容
l: $r1$ op $r2$ $r3$ → con	四則演算，比較演算，論理演算，構造データ操作等の2オペランド命令．($r1$: destination. $r2$, $r3$: operand)
l: $r1$ op $r2$ → con	四則演算，比較演算，論理演算，構造データ操作等の1オペランド命令．($r1$: destination. $r2$: operand)
l: $r1$ call f → con	解放されているレジスタファイルを1つ取り出し，そのレジスタファイル番号と関数名fを$r1$に格納する．
l: link i $r1$ $r2$	$r1$ が示している関数インスタンスの第i番目の receive 命令に$r2$ の内容（引数）を渡し，その receive 命令を起動する．
l: $r1$ rlink i $r2$ → con	$r2$ が示している関数インスタンスの第i番目の receive 命令に$r1$（レジスタ番号）を渡す．
l: $r1$ receive i → con	i番目の引数を受け取り$r1$ に格納する．
l: return $r1$ $r2$ → con	$r1$ に書かれているレジスタに$r2$ の内容を渡す．
l: $r1$ nins → con	解放されているレジスタファイルを1つ取り出し，そのレジスタファイル番号を$r1$ に格納する．
l: slink $r1$ $r2$ $r3$ → con	$r2$ が示しているレジスタファイルの$r1$ に$r3$ の内容を書き込む．
l: rins	この命令を含むインスタンスに割り付けられているレジスタファイルを解放する．
l: *sw* b → con_t, con_f	b の値が真なら con_t を継続命令とし，偽なら con_f を継続命令とする．
l: $r1$ move $r2$ → con	$r2$ の内容を$r1$ に書き込む．

注：各命令の最初の“l”はラベルである．
各命令の“→”に続く“con”はこの命令の継続命令である．



PE: Processing Element
SM: Structure Memory

図1 システムのアーキテクチャ
Fig. 1 System architecture.

の Datarol プロセッサ (PE) および構造メモリ (SM) で構成される．Datarol プロセッサは Datarol プログラムを実行する．Datarol プログラムのコードおよびオペランドデータはプロセッサ内のプログラムメモリ，データメモリに格納される．構造メモリは配列やリストなどの構造データを格納する．

2.2 プロセッサのアーキテクチャ

Datarol プロセッサは循環パイプライン・アーキテクチャの考えをベースに設計する．プロセッサは，演算部(FU)，リンクメモリ部(LM)，命令メモリ部(IM)，発火制御部 (AC)，オペランドメモリ部 (DM) を図2のように循環パイプライン状に結合して構成する．IM は Datarol コードを保持し命令フェッチを制御する．DM はレジスタ・ファイルでありオペランドデータを保持し，オペランドデータのフェッチと結果データの書き込みを制御する．FU は Datarol 命令の演算を実行する．LM は Datarol 命令間のリンケージ情報を保持し継続命令の IM 内アドレスの決定を行う．

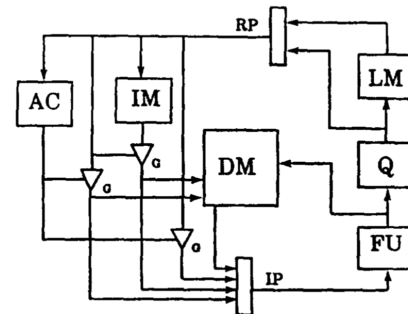


図2 プロセッサのアーキテクチャ
Fig. 2 Processor architecture.

3. Datarol プログラムの実行メカニズム

3.1 命令形式

Datarol マシンの命令セットを表1に示す．各命令の継続命令（"con"）とは各命令の次に実行される命令の集合である．ただし，ペアオペランドの存在チェックが必要な命令には記号 "*" が付加され，両方のオペランドの到着が確認されないと起動されない．図3（a）の Datarol プログラムをグラフで表現したものを図3（b）に示す．l_y と l_z は l_b が終了した後並列に起動され，l_a には記号 "*" が付加されているので，l_a は l_y と l_z の両方が終了した後起動される．

3.2 関数リンケージ

関数適用によって新たなインスタンスが生成されるとそのインスタンスは他のインスタンスと並列に実行される．呼び側のインスタンスの call 命令が起動されると新たなインスタンスにオペランドデータを保持するための作業領域（レジスタ・ファイル）が割り付けられる．関数本体（のコード）は，同一定義の関数本体を持つインスタンスで共有される．引数の受渡しは，呼び出し側のレジスタ・ファイルから新しく生成されたインスタンスのレジスタ・ファイルへデータを転送する link 命令でなされる．また新しく生成されたインスタンスの結果を書き込むべき呼び側のレジスタ名は，rlink 命令により新しいインスタンスへ送られる．link 命令で受け側へ引数と結果を書き込むレジスタ名が渡されると，それらに対応する receive 命令が起動され，receive 命令の継続命令へ実行の制御が移り，受け側の関数の実行が開始される．結果が生成されると受け側のインスタンスは return 命令で呼び側の rlink 命令で指示された結果を書き込むべきレジスタへ関数の結果を書き込む．return 命令の実行が終わると rins 命令によって受け側のインスタンスのレジスタ・ファイルは解放される．受け側の return 命令の実行後，呼び側では rlink 命令の継続命令へと実行の制御が移される．

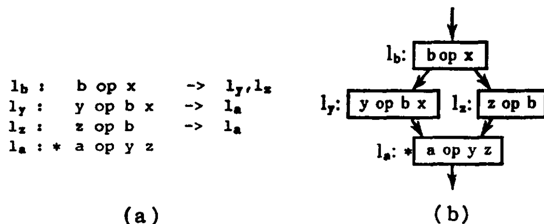


図 3 Datarol プログラムの例
Fig. 3 Example of Datarol program.

3.3 条件実行

条件実行の例として，図4（a）の条件式の部分に対する Datarol プログラムを図4（b）に示す（比較のため図4（c）にデータフローグラフも並べて挙げる）．図4（b）において，b の値が真ならば sw 命令の継続命令を l_{ii} とし，偽ならば継続命令を l_{jj} とする．この Datarol プログラムの例では，データフロー・プログラムに比べて sw 命令が必要となるが，3つの TF スイッチと2つの sink 命令が不用となる．

3.4 ループ実行

関数型言語における単一再帰式は，Datarol プログラムでは同期型か非同期型のいずれかのループで実行する[4]．同期型ループでは各繰り返しごとにループ変数が書き換えられた時点で次の繰り返しに入る．非同期型ループでは，nins 命令と slink 命令[4]により同じ関数本体を持つ新しいインスタンスを繰り返し作りループを実行していく．例えば，

```
function div(m, n : integer) return(integer)
  =for(x, y, z) initial(m, 0, n) do
    if x>z then
      {recur(u, v, z) where u=x-z,
                            v=y+1}
    else x;
```

という関数型言語 Valid のプログラムに対する Datarol プログラムは図5のようになる．slink 命令の継続命令は異なるインスタンスへの継続命令である．

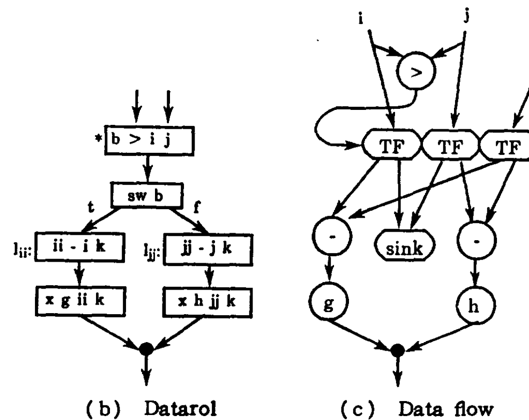
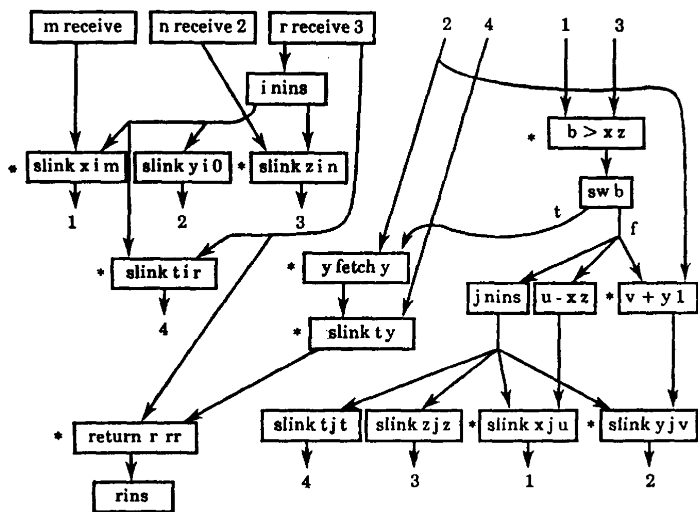


図 4 条件式の例
Fig. 4 Example of conditional expression.



(注) slink 命令が指す番号は図右上の番号への継続を示す

図 5 非同期型ループ
Fig. 5 Asynchronous loop.

4. Datarol プログラム抽出アルゴリズム

4.1 Datarol プログラムの抽出

関数型言語 Valid から Datarol プログラムを抽出するコンパイラの機能は，プログラムの Datarol プロセッサ上での実行速度や資源の利用効率に強く影響する．したがって，コンパイラは関数型言語に内在する並列性を最大限に引き出し，Datarol プロセッサの資源を有効に利用する Datarol プログラムを抽出することが望ましい．

ソースプログラムは，次の3つのステップを経て Datarol プログラムに変換される．

(A) ソースプログラムを中間言語表現に変換する．中間言語表現は，(w op u v) といった命令の集合であり[1]，データ依存関係を表す (op はオペレーション名，w は結果が入る変数，u, v はオペランド)．ここで，u, v, w はこの変換によって新しく生成された変数の名前である．

(B) ステップ(A)で得られたデータ依存関係からデータ依存解析により，マルチスレッド・コントロールフローを抽出する．

(C) それぞれの変数にレジスタを割り付ける．

このなかで，(A)に関しては従来の字句解析，構文解析等の方法で可能であるのでここでは述べず，以下(B)と(C)について述べる．

4.2 データ依存関係を表す概念

ここでは命令間の依存関係を解析するために必要となるいくつかの概念について述べる．以下，式 E を変換してできた命令の集合を node(E) で表す．また，命令 (w op u v) のことを単に w と表記する．

[命令間の直接依存関係：$<$]

ある命令集合 S の中に (p op q w) または (p op w q) という命令があったとすると，$p<q$ で表す．またその関係があるとき p を q の子，q を p の親と呼ぶ．

[命令間の依存関係：$\ll$]

命令間の依存関係を次のように定義する．

(1) すべての命令 p について，$p \ll p$

(2) 任意の命令 p, q, r について，$p<q$ かつ $q \ll r$ ならば $p \ll r$．

$p \ll q$ の関係がある場合，p を q の子孫 (descendant)，q を p の先祖 (ancestor) と呼ぶ．

[子孫 (Descendant)：Des]

$\mathrm{Des}(p)=\{q \mid q \ll p\}$.

[先祖 (Ancestor)：Anc]

$\mathrm{Anc}(p)=\{q \mid p \ll q\}$.

[子の共有子孫 (Common Descendant to all children)：CD]

$\mathrm{CD}(p)=\{q \mid \text{for all } w<p,\ q \in \mathrm{Des}(w)\}$.

ある命令 q が命令 p の子の共有子孫 CD(p) に含まれるということは，q の実行後，p を参照する命令がすべて終了していることを意味する．

また，CD(p) に含まれ CD(p) に先祖を持たないような命令を特に UCD (Upper CD) と呼ぶ．

$\mathrm{UCD}(p)=\{q \mid q \in \mathrm{CD}(p) \text{ and } \mathrm{Anc}(q) \cap \mathrm{CD}(p) = \{q\}\}$.

[占有先祖 (Proper Ancestor)：PA]

命令 q の子孫をどのようにたどっても命令 p を通るとき q は PA(p) に含まれる．

$\mathrm{PA}(p)=\{q \mid q=p \text{ or } w \in \mathrm{PA}(p) \text{ for all } w<q\}$.

また，PA(p) に含まれ，PA(p) に先祖を持たないものを特に UPA (Upper PA) と呼ぶ．

$\mathrm{UPA}(p)=\{q \mid q \in \mathrm{PA}(p) \text{ and } \mathrm{Anc}(q) \cap \mathrm{PA}(p) = \{q\}\}$.

[式の頭：head]

head(E) は，E に含まれる命令で，E の中に先祖を持たない命令の集合である．

$\mathrm{head}(E)=\{p \mid p \in \mathrm{node}(E) \text{ and } \mathrm{Anc}(p) \cap \mathrm{node}(E)$

$= \{p\}\}.$

［式の尾：tail］

tail(E) は，E に含まれる命令で，E の中に子孫を持たない命令の集合である．

$$\mathrm{tail}(E)=\{p \mid p \in \mathrm{node}(E) \text{ and } \mathrm{Des}(p) \cap \mathrm{node}(E) = \{p\}\}.$$

4.3 データ依存グラフからのマルチスレッド・コントロールフロー抽出

（1） 単純式 p の継続命令

p のすべての子 q について，p と q を直接結ぶパス以外のデータのパスが存在しないならば，w を p の継続命令とする．p の継続命令 cont(p) の定義を以下に示す．

$$\mathrm{cont}(p)=\{q \mid q<p \text{ and } \mathrm{not}(q \ll w) \text{ for all } w<p,\ w \neq q\}.$$

このことによって，データフローでは必要であるが，Datarol プログラムでは必要ないパスを除去し，ペアオペランドマッチング数を減らすことができる（図6）．

（2） 条件式 p の継続命令

p が

if b then E_t else E_f

という形をしているとし，

$\mathrm{tail}(E_t)=m,\ \mathrm{tail}(E_f)=n$

であるとすると，b が真の場合の p の継続命令は UPA(m) であり，偽の場合の p の継続命令は UPA(n) である．このようにして継続命令を決定すると，条件によっては実行しても無駄となる命令を実行せずにすむ．

（3） call 命令の起動

call 命令のオペランドは関数の名前のみであり変数ではないので（1）で述べた単純式の継続命令とはならない．そこで call 命令は，実行されることが分かった時点で起動することとする．つまり，call 命令が true part または false part に含まれるならばその条件分岐命令（sw）の継続命令の要素とする．それ以外の場合は，その call 命令を含む関数の receive 命令のうちのどれかひとつの継続命令の要素とする．

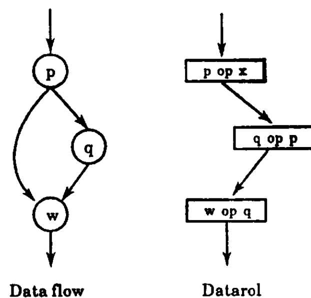


図6　パスの除去
Fig. 6 Removing pass.

（4） ループ構造の Datarol コード抽出

単一再帰式を含む関数型言語の関数を中間言語表現に変換する際，再帰式を呼び出す側の関数（以下呼び側という）と再帰式に相当する関数（以下受け側という）のふたつの関数が生成される．Datarol プログラムでは同期式，非同期式いずれの方式でもそれらはひとつの関数で表される．以下，中間言語表現から両方式のプログラムを生成する方法を述べる．

〈同期型ループ〉

（1） 受け側，呼び側の call 命令 q を継続命令として持つすべての命令 p について，p の継続命令は

$\mathrm{cont}(p)-q+R$

とする．ただし，R は q の継続命令である link 命令の集合．

（2） 受け側，呼び側のすべての link 命令 q に対して，

link n i $x \rightarrow t$ move x

と書き換える．ただし，t は link 命令に対応する receive 命令の結果を書き込む変数名．

（3） （2）で書き換えてできた各 move 命令 p について，p が x（x は p の結果を書き込む変数）の生存区間（4.4 節参照）に含まれるならば以下の操作を行う．Q と R の和集合を S とし，S に含まれるすべての要素の継続命令に p を加える．ここで，Q は p を継続命令として持つ命令の集合，R は x をオペランドとして持つ命令で p の先祖でない命令の集合である．ただし，S の要素が3つ以上ある場合には，NOP 命令を用いて同期をとる．以上により，ループ変数の更新は，そのループ変数の利用がすべて終了した後にすることができる．

（4） （2）で生成された各 move 命令 p について，p の元の link 命令に対応する receive 命令の継続命令を p の継続命令とする．

（5） 受け側の return 命令を継続命令として持つ各命令 p（p は receive 命令でないもの）について，呼び側の rlink 命令の継続命令を p の継続命令とする．

（6） 呼び側の rlink 命令の結果を書き込む変数名を受け側の対応する return 命令の第2オペランド

(呼ばれた関数の結果が入る変数) に書き換える.

(7) 受け側の return, rins, 呼び側と受け側の rlink 命令, およびいずれの命令の継続命令にもならない命令を削除する.

(8) ループの最適化 (不要な命令, 変数の除去等) を行う.

(9) レジスタ割付けを行う. (4.4 節参照)

〈非同期型ループ〉

(1) ふたつの関数それぞれについて (1)-(3) の方法で次命令を決定し, それぞれのレジスタ割付け (4.4 節参照) を行う. ただし, 呼び側のレジスタ名と受け側のレジスタ名が重複しないように割り付ける.

(2) 受け側, 呼び側の call 命令を nins 命令に, link 命令, rlink 命令を slink 命令に書き換える.

r_i call 'function →r_i nins

link m r_i r_j　　→slink r_k r_i r_j

r_s rlink n r_i　　→slink r_l r_i r_s

ただし, r_k, r_l はそれぞれの link, rlink 命令に対応する receive 命令の結果を格納するレジスタ.

(3) (2)で書き換えてできた各 slink 命令 p について, 元の link, rlink 命令に対応した receive 命令の継続命令を p の継続命令とする. なお, rlink 命令であった p の継続命令には受け側の rlink であった slink 命令を加える.

(4) (2)で書き換えてできた受け側の slink 命令がすべて終了したらそのインスタンスを解放するように rins 命令を挿入する.

(5) 受け側の return 命令を,

return r_i r_j→slink r_i r_j

と slink 命令 p に書き換え, Q と $\{r\}$ の和集合を p の継続命令とする. ただし Q は呼び側の rlink 命令の継続命令, r は受け側の rins 命令.

(6) 受け側の rlink 命令であった slink 命令について,

slink r_i r_j r_k→slink r_i r_j r_l

と書き換える.

(7) どの命令の継続命令にもなっていない命令を削除する.

Datarol プログラムにおける同期型, 非同期型の選択は, ループ変数のループフェーズ間でのデータ依存性を調べて行う. ここで, あるループフェーズで定義されたループ変数の値が次のフェーズの計算で必須[8]となるとき, その変数はループフェーズ間でデータ依存性を持つという. 非同期型のループではデータ依存性を持つループ変数が少ないほどループフェーズ間の並列実行の効果が得られるが, nins 命令を用いるので新しいインスタンスを生成するためのオーバヘッドが生じる. 一方同期型のループではインスタンス生成のオーバヘッドは生じないが, ループフェーズごとに同期をとるのでループフェーズ間の並列実行は行えない. したがって, データ依存性を持つループ変数がある数以下ならば非同期型ループのコードを生成するようにする.

4.4 レジスタ割付けの最適化

個々の関数に必要なレジスタ数はレジスタ・ファイルを有効に利用するためにできるだけ少なくすべきである. ある変数に割り付けられたレジスタはその変数を参照する命令の実行がすべて終了したら解放することができる. そして, そのレジスタは新たな変数に割り付けることができる. レジスタを解放できるか否かの判定は, 変数の生存区間を用いて行う.

変数 x の生存区間 $Iv(x)$ は 4.2 節で定義した UCD の概念を用いて, 区間

$$[\mathrm{birth}(x), \mathrm{death}(x)]$$

$$\text{ただし } \mathrm{birth}(x)=lx,$$

$$\mathrm{death}(x)=\mathrm{UCD}(x).$$

と定義する. ここで, lx は x を結果とする命令のラベルである. 変数 x に割り付けられているレジスタは x の生存区間では解放することはできないが, 生存区間以外では他の変数に再割付けすることができる. もし, 変数 x の最後の参照がある条件分岐命令の true part E_t あるいは false part E_f で行われた場合には death(x) は, E_t, E_f 両方で定義される. この場合は death(x) は,

$$\mathrm{death}(x)=(\mathrm{UCD}(x)\cap E_t\,;\ \mathrm{UCD}(x)\cap E_f).$$

とする. 生存区間の一例を図 7 に示す. birth(x) は lx であり, death(x) は $(lc\,;ld)$ である. したがって, $Iv(x)$ は, 区間 $[lx, (lc\,;\,ld)]$ となる.

この生存区間を表す式はレジスタ割付けの際 true part と false part に分けて適用される.

例えば, 生存区間が

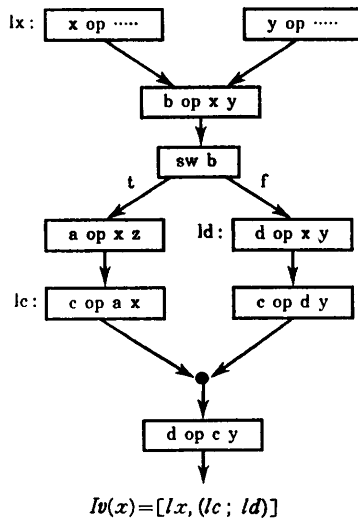
$$[a, (b, c\,;\ d, e, f)]$$

となっていれば true part と false part それぞれの解析で, $[a, (b, c)]$ と $[a, (d, e, f)]$ という生存区間が用いられる.

次にレジスタ割付けのアルゴリズムを示す.

(1) node(E) にある全変数の生存区間を求める.

(2) 以下のように RS, $Rvar$, Var を求める.



$Iv(x)=[lx, (lc\,;\, ld)]$

図 7 *Iv* の例
Fig. 7 Example of *Iv*.

RS := {すべてのレジスタの集合},

$Rvar$:= {node(E) にあるすべての変数},

Var := head(E).

(3) $Rvar := Rvar - Var$ とし，Var 中のそれぞれの変数に異なるレジスタを割り付ける．R をここで割り付けられたレジスタの集合であるとして，

$RS := RS - R.$

とする．

(4) Var 中のある y に対して，$lx<$death(y) (lx は命令 x のラベル) であるような $Rvar$ の要素 x が存在すれば y に割り付けられているレジスタを x に割り付ける．

$Rvar := Rvar - \{x\},$

$Var := Var \cup \{x\} - \{y\}.$

(5) ステップ(4)の条件を満たすレジスタがなければ，RS 中にあるレジスタ r_j を $Rvar$ 中にある x に割り付ける．

$Rvar := Rvar - \{x\},$

$Var := Var - \{x\},$

$RS := RS - \{r_j\}.$

(6) $Rvar$ が空集合になるまで(4)と(5)を繰り返す．

一般に，以上のアルゴリズムによりレジスタ割付けを行うと必要なレジスタは中間言語表現に含まれていた変数より少なくなり Datarol プロセッサの DM 内のレジスタを有効に利用することができる．

● 同期型ループのレジスタ割付け

同期型のループ本体ではレジスタは解放しない．その理由はレジスタを解放すると以下のような問題が生じるからである．もし各ループフェーズで値が定義される変数 x に割り付けられていたレジスタ r を解放し別の変数 y に割り付けたとすると，x の値（格納されているレジスタは r）を次のフェーズのために更新した後に y（レジスタ r）の参照が起こった場合誤りとなる．このような問題は変数の値の更新を行う前に同期をとることによって解決できるが，この場合，同期のためのオーバヘッド（命令数と待ち合わせの時間）が増えるので好ましくない．

5. Datarol プログラムの評価

この章では簡単な Valid のプログラムについて，上記のアルゴリズムを用いて抽出された Datarol プログラムと，そのデータフロー・プログラムとの比較について述べる．ここで比較の対象とするデータフロー・アーキテクチャは NTT の DFM[5]であるが，マンチェスタ大学のデータフローマシン[9]，MIT の ID[6]，電総研の Sigma-1[10]も同様の条件分岐，関数呼び出し，ループ実行の制御機構を持つので同様の比較ができると考えられる．また，Datarol プログラムと手続き型プログラムのコンパイルコードとを比較するために，同様のプログラムを C 言語で手続き的に書き，それを 68020 用コードにコンパイルしたものとの比較も行う．

要素 a が集合 b に含まれるなら true(1), 含まれないなら false(0) を返す関数 member は関数型言語 Valid で図 8(a)のように書ける．これをデータフロー・プログラムで表したものを図 9，Datarol プログラムで表したものを図 10 に示す．

```
function member (a, b: list) return (bool)
=if null(b) then false
  elseif a=car(b) then true
      else member(a, cdr(b));
```

(a) Valid

```
(defun member (a b)
   (if (null b) nil
      (if (=a (car b)) t
      (member a (cdr b)))))
```

(b) LISP

```
member (a, b){
   while (!(null(b))){
      if (a==car(b)) return 1;
      else b=cdr(b)}}
```

(c) C

図 8 member のプログラム
Fig. 8 Programs of member.

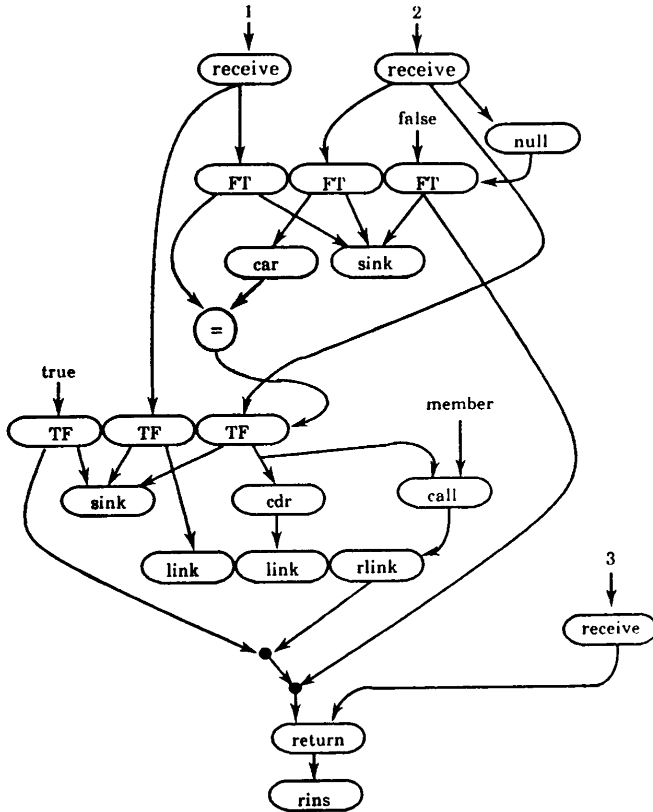


図 9 member のデータフロー・プログラム
Fig. 9 Data flow program of member.

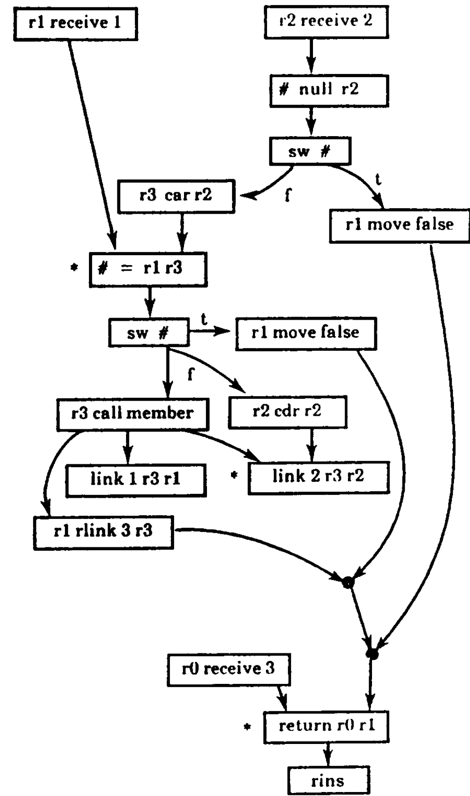


図 10 member の Datarol プログラム
Fig. 10 Datarol program of member.

ノード数はデータフロー・プログラムが 25, Datarol プログラムが 17, ペアオペランドマッチング数はデータフロー・プログラムが 8, Datarol プログラムが 3 でともに Datarol プログラムの方が少なくなっている．ただし，データフロー・プログラムには sink 命令に伴うゴミ・トークンの処理のための (同期の) 命令を含めていないので，実際はこのデータよりさらに大きい値となる．Datarol プログラムではレジスタを用いるのでこのような処理を必要としない．また，Datarol プログラムについてはレジスタ割付けを行った結果，中間言語表現における 7 個の変数に対して必要なレジスタは 4 個となる．

member を Lisp で図 8 (b) のように書き，HCL (Hokkaido Common Lisp)[11] のコンパイラにかけるとそのオブジェクトコードの命令数は 26 となる．

また，member を C で図 8 (c) のように書くと (型定義省略)，そのコンパイルの結果は，**図 11** のようになる (日立製作所 E-7300, C コンパイラ)．

図 11 のコードの命令数は 24 である．しかし，Datarol およびデータフロー・プログラムでは *null*, *car*, *cdr* をプリミティブな命令として扱っているので，C における *null*, *car*, *cdr* のサブルーチン呼び出しのためのスタック操作等 (図 11 に * で示してある) を除いて比較しなければならない．よって，図 11 の命令数は 19 とカウントされる．

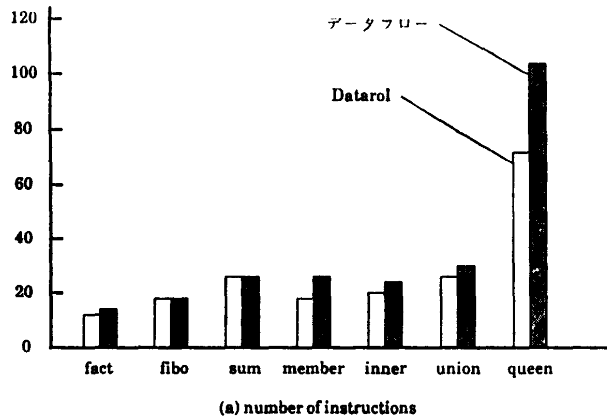
このほかにも，集合演算，N-Queen，行列の演算，フィボナッチ数列，階乗，総和等を求める問題を解く Valid のプログラムから Datarol コードを抽出し，それぞれのプログラムに対応するデータフロー・プログラムとの命令数 (**図 12**(a))，ペアオペランドマッチング数 (図 12(b)) について比較を行った．また，68020 コード (ソース言語 : Lisp, C) との命令数 (**図 13**(a))，必要レジスタ数 (図 13(b)) の比較を行った．

〈データフロー・プログラムとの比較〉

図 14(a) にデータフローの全命令数に対する TF スイッチ数の割合とデータフローに対する Datarol の命令数の比の関係を示す．データフロー・プログラムの TF スイッチの割合が比較的小さいプログラムはほぼ同等，TF スイッチの割合が大きいプログラム

```
_member
     link      fp, $-4
     jra       L 86
L88:
     move. 1   12(fp), -(sp)
     jsr       _car
    *add. 1    $4, sp
     move. 1   d 0, -4(fp)
     move. 1   8(fp), d 0
     cmp. 1    -4(fp), d 0
     jne       L 89
     move. 1   $1, d 0
L83:
     unlk      fp
     rts
L89:
     move. 1   12(fp), -(sp)
     jsr       _cdr
    *add. 1    $4, sp
     move. 1   d 0, -4(fp)
     move. 1   -4(fp), 12(fp)
L86:
    *move. 1   12(fp), -(sp)
     jsr       _null
    *add. 1    $4, sp
    *tst. 1    d 0
     jeq       L 88
     clr. 1    d 0
     jra       L 83
```

図 11 member のマシンコード (ソース: C)
Fig. 11 Machine code of member (source: C).



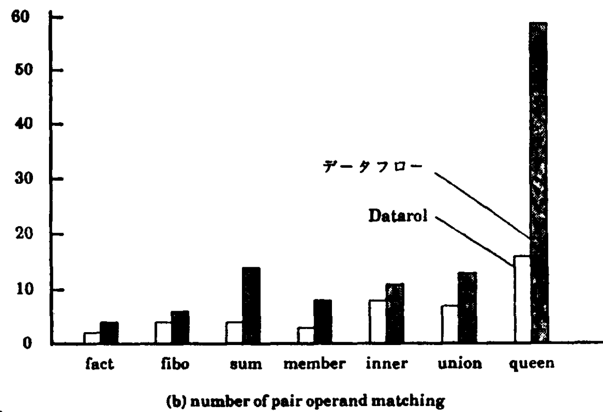


図 12 Datarol とデータフローの比較
Fig. 12 Comparison of Datarol with Data flow.

ほど大きな改善が見られる．また，図 14(b)にデータフローの全命令に対する TF スイッチの割合とデータフローに対する Datarol のペアオペランドマッチング数の比の関係を示す．ペアオペランドマッチング数も TF スイッチの割合が大きいほど改善が見られるが，TF スイッチが少ないプログラムでも図 6 に示したようなパスの除去により改善が認められる．

〈Lisp のコンパイルコードとの比較〉

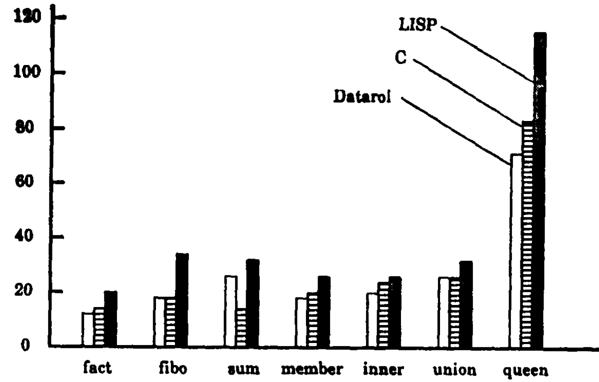
すべての関数について Datarol プログラムの方が命令数が少なくなっている (図 13(a))．これは，関数型プログラムを逐次型のコードにコンパイルすると関数呼び出しの際のスタック操作が必要となるためである．また，レジスタ数はほぼ同程度になっている (図 13(b))．

〈Cのコンパイルコードとの比較〉

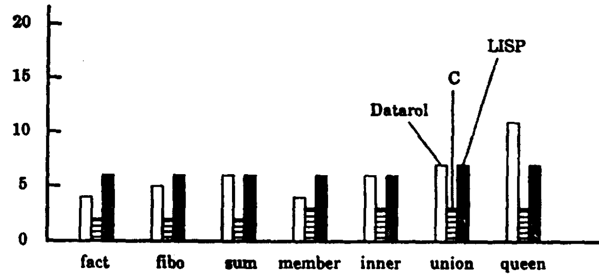
図 13(a)を見ると sum, queen を除きほぼ同程度の命令数になっていることが分かる．ここで，sum はCでは繰り返しを用いているが Datarol プログラムでは分割統治法を用いているため Datarol プログラムの命令数が多くなっている．また，queen はCでも再起呼び出しを用いているので，そのスタック操作のためにCの命令数が多くなっている．また，図 13(b)を見るとレジスタ数は，2～3倍程度であり，命令数が2桁の Datarol プログラムについては 16～32 個程度のレジスタを持つプロセッサを想定すると現実的な数字であると考えられる．

6. おわりに

本論文では，データ依存関係によって，関数型言語 Valid から Datarol マシンの機械語である Datarol コードを抽出するコンパイル法について述べ，いくつかのサンプル・プログラムについてその評価を行った．Datarol マシンは，データフロー制御や発火制御のオーバヘッドなど，データフローマシンの持つ欠点を取り除き，効率的な超多重並行処理の制御を行うことを特徴とする．Datarol コードはデータフロー・プログラムからペアオペランドの存在チェックやゲート演算などの余分なデータフロー制御を取り除いて最適化したマルチスレッド・コントロールフロー・プログ



(a) number of instructions



(b) number of registers

図 13 Datarol と C, LISP の比較
Fig. 13 Comparison of Datarol with C and LISP.

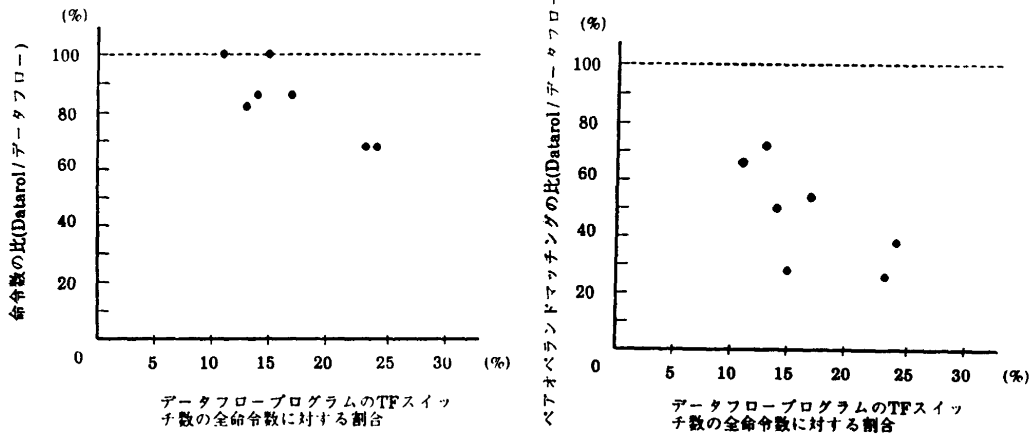


図 14 TF スイッチ数と改善度の関係
Fig. 14 Relation between number of TF switch and degree of improvement.

ラムである. 本論文ではまず, データ依存解析に基づくコンパイルのアルゴリズムを示し, そのアルゴリズムによって得られた Datarol コードをデータフロー・プログラムと比較した. 比較は演算やスイッチなど実行時間に影響を与えるノードの個数, ペアオペランドマッチング数, また実行に必要なメモリ／レジスタ数について行い, Datarol プログラムがデータフロー・プログラムの欠点を改善するものであることを示した.

また, 同じサンプル・プログラムを LISP および手続き型言語Cで記述してその逐次マシン (MC 68020) のコンパイルコードを Datarol コードと比較した.

その結果，これらのサンプル・プログラムに関して，Datarol コードの命令ノード数が Lisp およびCのコンパイルコードの命令ステップ数とほぼ同程度であることが分かった．

このことは Datarol マシンが高い実行性能を持つ可能性があることを示している．ただし，これらの議論をより一般的なものとするためにはさらに多くの例題プログラムについて比較データを積み重ねていく必要がある．今後はこのアルゴリズムに基づいたコンパイラを完成させ，より多くの実用的なプログラムについてアルゴリズムおよび Datarol プログラムの評価を行っていく．

参考文献

1) 雨宮：超多重並列処理のためのプロセッサ・アーキテクチャ，情報処理学会コンピュータアーキテクチャシンポジウム，pp. 99-108 (1988).
2) 上田，谷口，雨宮：Datarol プロセッサのアーキテクチャについて，第 38 回情報処理学会全国大会論文集，pp. 1406-1407 (1989).
3) 雨宮，長谷川，小野：データフロー計算機用高級言語 Valid，研究実用化報告，Vol. 33, No. 6, pp. 1167-1181 (1984).
4) 長谷川，雨宮：データフローマシン用関数型高級言語 Valid，電子情報通信学会論文誌，Vol. J 71-D, No. 8, pp. 1532-1539 (1988).
5) Amamiya, M., Takesue, M., Hasegawa, R. and Mikami, H.: Implementation and Evaluation of List-Processing-Oriented Data Flow Machine, *Proc. 13th ISCA*, pp. 10-19 (1986).
6) Arvind, Gostelow, K. P. and Plouffe, W.: An Asynchronous Programming Language and Computing Machine, Tech. Report, TR-114 a, Univ. California, Irvine (1987).
7) SISAL: Streams and Iteration in a Single Assignment Language, Language Reference Manual version 1.2, M-146, Report of Co-operation of the Colorado State University, DEC, Lawrence Livermore National Laboratory and University of Manchester (1984).
8) Ono, S., Takahashi, N. and Amamiya, M.: Optimized Demand Driven Evaluation of Functional Programs on a Dataflow Machine, *1986 Int. Conf. of Parallel Processing*, pp. 421-428 (1986).
9) Gurd, J., Kirkham, C. C. and Watson, I.: The Manchester Prototype Dataflow Computer, *Comm. ACM*, Vol. 28, No. 1, pp. 34-52 (1985).
10) Shimada, T., Hiraki, K. and Nishida, K.: An Architecture of a Data Flow Machine and Its Evaluation, *Proc. COMPCON84 Spring, IEEE*, pp. 486-490 (1984).
11) 山本：北海道大学コモン・リスプ (HCL) リリース・ノート，version 1.0 (1988).

(平成元年6月 1 日受付)
(平成元年9月 12 日採録)

立花　徹（学生会員）
昭和 40 年生．昭和 63 年九州大学工学部情報工学科卒業．現在，同大学院総合理工学研究科修士課程在学中．並列処理用言語の研究に従事．

谷口倫一郎（正会員）
昭和 30 年生．昭和 53 年九州大学工学部情報工学科卒業．昭和 55 年同大学院工学研究科修士課程修了．同年九州大学大学院総合理工学研究科情報システム学専攻助手．平成元年より同助教授．工学博士．画像理解，画像処理システム，並列処理システムの研究に従事．電子情報通信学会，人工知能学会各会員．

雨宮　真人（正会員）
昭和 17 年生．昭和 42 年九州大学工学部電子工学科卒業．昭和 44 年同大学院工学研究科修士課程修了．同年日本電信電話公社武蔵野電気通信研究所入所．以来，プログラミング言語・処理系，自然言語理解，データフロー・アーキテクチャ，並列処理，関数型／論理型言語，知能処理アーキテクチャ，等の研究に従事．現在九州大学大学院総合理工学研究科情報システム学専攻教授．工学博士．電子情報通信学会，ソフトウェア科学会，人工知能学会，IEEE，AAAI 各会員．