述語付きソフトウェア・パイプラインへのSpiral Graphによるレジスタ割付け

糸賀裕弥[†,☆] 山下義行[††] 田中二郎[†††]

ソフトウェア・パイプラインによる最適化をハードウェアで補助する仕組みとして，レジスタ改名機構と述語付き命令実行機構がある．レジスタ改名機構向けのレジスタ割付け手法として Spiral Graph を用いた方法が提案されているが，述語付き命令実行機構を備えた場合の割付け方法について詳しい解析がなされていなかった．本論文では，それら 2 つの補助機能を備えた IA-64 のようなアーキテクチャに対するレジスタ割付け手法として，述語付き Spiral Graph を提案する．述語付き Spiral Graph は，従来のトラックを述語の値に応じた副トラックの集合と見なすことで，述語が異なる命令で別に定義される生存区間を適切に表現することができる．述語付き Spiral Graph を用いることで，必要レジスタ数を最小とするレジスタ割付けが多項式時間で行える．提案するアルゴリズムでは，必要レジスタ数の下界である $W_{\max}$ 本のレジスタで割付けが行えるか判定を行い，必要レジスタ数が $W_{\max}$ 本の場合の割付け結果が得られ．そうでない場合には，$W_{\max}+1$ 本での割付け結果が得られる．

Register Allocation for Predicated Software Pipelining Using Spiral Graph

Hiroya Itoga,[†,☆] Yoshiyuki Yamashita[††] and Jiro Tanaka[†††]

Register renaming and predicated execution are the hardware facilities to support software pipelining. Spiral Graph has been proposed to allocate registers for the architectures with register renaming. However, the original Spiral Graph cannot naturally express the predicated execution. In this paper, authors propose Predicated Spiral Graph for the architectures that support software pipelining, such as Intel IA-64 architecture. Predicated Spiral Graph extends a track on the original Spiral Graph to multiple sub-tracks, each of which represents each value of the predicates. The live ranges, which are defined and used in the predicated execution, are allocated to the sub-tracks. Authors propose the algorithm on Predicated Spiral Graph, which yields the optimal number of the required registers in polynomial time. This algorithm judges whether the number of the required registers equals to $W_{\max}$, a lower boundary of them, and yields a result of allocation. Even if the check fails, the algorithm yields a result which requires $W_{\max}+1$.

1. はじめに

最適化コンパイラにおけるレジスタ割付けの問題は，プロセッサの実行性能が高くなるほどに重要になってきている．レジスタはプロセッサ内部で高速に動作するメモリの一種であり，プログラム中の変数や，演算の途中結果などをレジスタに格納することで高い実行性能を得られる．しかし，プロセッサの物理的な制限や命令セットの制限などにより，レジスタの容量は制限されている．このような容量の少ないレジスタに対して適切なレジスタ割付けを行い，データの主記憶への退避・回復のための処理を減少させることで，高い実行性能を保持することができる．

ソフトウェア・パイプライン[1)]は，命令レベル並列性を用いた単体プロセッサ向けのループ最適化方法として一般的に普及している．ソフトウェア・パイプラインでは，ループプログラム中の繰返しを一定間隔でずらして重ね合わせるようにコンパイル時に命令を並べ換えることで，ある繰返しにおける演算あるいは

† 筑波大学工学研究科
Doctoral Program in Engineering, University of Tsukuba
†† 佐賀大学理工学部
Department of Information Science, Saga University
††† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba
☆ 現在，茨城県工業技術センター
Presently with Industrial Technology Institute of Ibaraki Prefecture

```
do {
0:  load  v1
1:  add   v6, v1 -> v4
2:  mult  v4, v4 -> v5
3:  add   v5, v1 -> v2
4:  mult  v1, v2 -> v3
5:  add   v2, v3 -> v6
6:  store v2
   slide +1
}
```

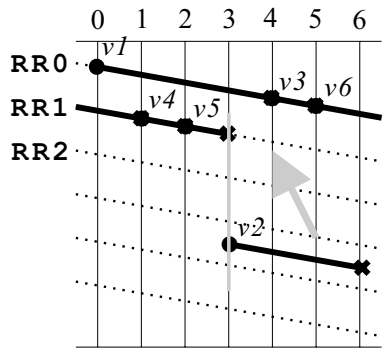

図 1 Spiral Graph
Fig. 1 Spiral Graph.

メモリ参照による実行待ち時間に，その命令とデータ依存のない命令，すなわち別の繰返しの命令を実行することでループの総実行時間を短くすることが可能である．

ソフトウェア・パイプラインはその実行方式から，繰返し開始間隔（Initiation Interval: II）よりも長く同一の変数を保持できない，条件分岐を含むプログラムへの適用が難しいという 2 つの制限を受ける．この制限に対して，前者の問題にはレジスタ改名機構が，後者の問題には述語付き命令実行機構がハードウェア補助機構として提案された[3]．現在，普及が始まっているインテルの IA-64 アーキテクチャ[9] においては，レジスタ改名機構としてローテイティング・レジスタ（rotating register）が，述語付き命令実行機構としてプレディケーション（predication）が実装されており，これに対するレジスタ割付けが重要な問題となっている[4]．

このようなレジスタ改名機構向けのレジスタ割付け方法として，Spiral Graph[8] が提案されている．Spiral Graph はローテイティング・レジスタの特徴であるレジスタの改名幅が 1 の場合には，多項式時間での最適レジスタ割付けが可能であるなどの利点があるが，述語付き命令実行機構を備えた場合のレジスタ割付け方法についての解析がほとんどなされていなかった．

本論文では，レジスタ改名機構と述語付き命令実行機構を備えたアーキテクチャにおいて，条件分岐を含むプログラムをソフトウェア・パイプライン化した場合の，Spiral Graph を用いたレジスタ割付け方法を提案する．

2 章では基礎知識として，レジスタ改名機構と Spiral Graph および述語付き命令実行機構について述べる．3 章において，Spiral Graph に対する拡張である述語付き Spiral Graph を提案し，4 章において，述語付き Spiral Graph における必要レジスタ数の下界である $W_{\max}$ を定義する．5 章では，$W_{\max}+1$ 本以下となる多項式時間でのレジスタ割付けアルゴリズムを提案し，必要レジスタ数が $W_{\max}$ 本すなわち最小となる条件を述べる．

2. 基礎知識

2.1 レジスタ改名機構と Spiral Graph

本論文で考えるレジスタ改名機構は，レジスタに保持されている値を変更せずに，プロセッサが持つ多数のレジスタの名前であるレジスタ番号をいっせいに +1 だけ変化させることのできるハードウェア機構である．すなわち，k 番目のレジスタに保持されている値は，レジスタ改名後に $k+1$ 番目のレジスタの値として利用できることを意味する．

レジスタ改名機構に対するレジスタ割付け方法として Spiral Graph を用いた方法が提案されている[8]．Spiral Graph では，レジスタ改名によるレジスタ名の変化を傾いた線によって表現する．**図 1** に Spiral Graph の例を示す．グラフの縦軸は実レジスタ番号，横軸はプログラム中の命令ステップ番号である．ループの表現のためにグラフの左右は連続している．つまり，v_6 はレジスタ`RR0`に対して定義されたのち`RR1`として参照される．`slide +1`はレジスタ改名のための命令である．Short Bridge Algorithm は Spiral Graph 上に隙間が小さくなるように生存区間を巻き付けていく，レジスタ割付けアルゴリズムである．v_2 は，v_5 との隙間が小さくなるようにらせんに巻き付けられる．

レジスタ割付けとソフトウェア・パイプラインにおける命令スケジュールは互いに関連しており，一般に命令スケジュールを先に行った方が命令の並列度を大きくできる[12]．そこで，本論文においてもあらかじめスケジュールされたプログラムに対してレジスタ割付けを行うこととする．

一般のループプログラムにおけるレジスタ割付け問

```
DO I=1, 1000
  IF ((( A(I) + PI )*2.0 ) > 0.0 )
    THEN
      Y(I)=(( X(I)+1.0 )**2.0 + X(I) + 1.0 ) * 3.0
    ELSE
      W(I)=U(I)**2 + 4.0
  END IF
END DO
```

図 2 サンプルプログラム
Fig. 2 A sample program.

```
L: (p17) ldfd f(d)=X[I+3]              : M(3*II+0)
   (p19) fma  f(f)=f(e)*f(e)+f(e) ;;   : F(5*II+0)

         ldfd f(a)=A[I]                : M(0*II+1)
         fmp  f(c)=f(b)*2          ;;  : F(2*II+1)

   (p33) ldfd f(h)=U[I+3]              : M(3*II+2)
   (p18) fadd f(e)=f(d)+1            ;;  : F(4*II+2)

   (p20) stfd Y[I+6]=f(g)              : M(6*II+3)
         fadd f(b)=f(a)+PI           ;;  : F(1*II+3)

   (p35) stfd W[I+5]=f(i)              : M(5*II+4)
   (p34) fma  f(i)=f(h)*f(h)+4       ;;  : F(4*II+4)

   (p19) fmpy f(g)=f(f)*3            ;;  : F(5*II+5)

         fcmp.gt p16,p32=f(a),0        : F(2*II+6)
         br.ctop L                   ;;  : B
```

図 3 スケジュール例
Fig. 3 A result of instruction scheduling.

題は，NP 完全問題であるが，Spiral Graph においてレジスタ改名幅が 1 の場合には，埋め草（fictitious interval）を用いた Short Bridge Algorithm の拡張により，生存区間の総数を N としたとき $O(N^2)$ の多項式時間で，レジスタ数が最小となる割付け結果が得られることが示されている[7]．

2.2 述語付き命令実行機構

述語付き命令実行機構は，条件分岐を含むプログラムを IF 変換[2] することでプロセッサの命令ごとに述語（predicate）をあらかじめ付加し，ハードウェアにより述語の値に応じて命令の実行・非実行を選択できるハードウェア機構である．もとのプログラムの条件判定命令は述語の設定命令へと変換され，この値が真であった場合には命令は実際に実行されるが，偽であった場合には`NOP` 命令と同様に実行結果に影響を与えない命令と見なされる．

2.3 サンプルプログラム

本論文では，説明のために**図 2** で示されるサンプルプログラムを用いる．プログラムをコンパイルした結果の一例を**図 3** に示す．

スケジューリングには以下の条件を仮定した．プロセッサはメモリ操作命令を 1 つ，浮動小数点数演算命令を 1 つ，そして整数演算または分岐命令のうち 1 つ，計 3 命令を同時に発行可能である．命令の前に付加されている`(p17)` などが述語レジスタを示す．`ldfd`, `stfd` は浮動小数点数のロード，ストア命令であり，メモリからの値のロードには 9MC（マシンサイクル）を必要とする．`fma`，`fmpy`，`fadd` は浮動小数点数の積和，積，和演算であり，結果が確定するまでに 5MC を必要とする．`fcmp` は述語の設定命令であり，この場合，条件が満たされた場合には`p16` が真となり，`p32` が偽となる．逆に，条件が満たされなかった場合には`p16` が偽となり，`p32` が真となる．`br.ctop` 命令はループの終了判定，レジスタ改名，ループバックジャンプを 1 命令で行えるとする．これらは 1MC を要する．なお，`f(a)` などは浮動小数点数レジスタを表現してお

り，述語レジスタとともに +1 だけレジスタ改名される．1，2，PI などの値は，あらかじめ特定の浮動小数点数レジスタに値が保持されている．ロードする値の添字は，それぞれの命令と同時に発行される整数演算命令によって適当に調整される．

命令の右側に付したのは，命令の種類，ステージ番号と実行スロット番号である．II は繰返し開始間隔（Initiation Interval）であるから，M(3*II+0) は，3 ステージ目の 0 番目の命令スロットにおいてメモリ命令が発行されることを示す．

ここで仮定したアーキテクチャは，インテル IA-64 において MFI/MFB の命令バンドルを用いた場合と似ていることに注意しておく．

3. 述語付き Spiral Graph

筆者らは，レジスタ改名機構向けの Spiral Graph に対して，述語付き命令実行機構への拡張を行った述語付き Spiral Graph を提案している[11]．**図 4** は述語付き Spiral Graph の例である．

述語付き命令実行機構を持つアーキテクチャにおいては，条件分岐による複数の実行パスを，1 つの実行パスの命令列として表現する．実行時には，述語の値を参照し，命令ごとに実行・非実行を選択しつつ演算が行われるため，条件判定が真の場合に行われる生存区間の定義や参照と，偽の場合に行われる生存区間の定義や参照が 1 つの命令列に混在していることになる．

プログラム中で同時に存在する変数の生存区間どうしを，同じ実レジスタに割り付けることは通常できない．しかし，述語付き命令実行機構を持つアーキテクチャにおいては，プログラム上は同時に存在している生存区間であっても，条件判定の結果が真の場合に定義，参照される変数と，偽の場合に定義，参照される変数を同じ実レジスタに割り付けることができる．これは，条件判定の結果が同時に真かつ偽であることはありえないからである．

そこで筆者らは従来の Spiral Graph を拡張し，プログラム上は同時に存在するが同時に値を保持することがない，述語が互いに異なる生存区間を同じ実レジスタに割り付けることを，図 4 における true あるいは false の副トラック（sub-track）で表現した☆．副トラックはそれぞれ特定の述語の状態を表現しており，対応する述語の値で定義，参照される生存区間だけが，該当する副トラックに割り付けられる．述語の数が増えた場合には，副トラックの数を増やす．述語を考慮して複数の生存区間を 1 つの実レジスタに割り付けることを実レジスタを「共有する」と呼ぶ．

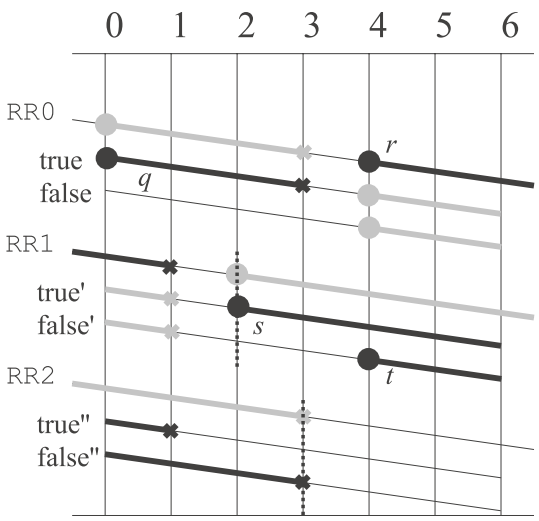

図 4 述語付き Spiral Graph
Fig. 4 Predicated Spiral Graph.

述語の値は条件判定命令を実行するたびに更新される．ソフトウェア・パイプラインによる実行では，複数の繰返しが重ね合わせられるように実行されるため，パイプライン・ステージごとに述語の値は異なる．生存区間ごとにパイプライン・ステージ番号の情報を付加することも可能だが，ここではステージ数の変化の情報を「'」の数で表現し，true' や false" として区別している．

従来のトラックは副トラックの集合として表現されるため，図 4 の q のように条件判定の結果が真の場合のみの生存区間も従来の生存区間として扱える．逆に，r は述語によらず定義，参照される生存区間であるが副トラックをすべて占有するとして定義する．また，s と t に示されるように，1 つの実レジスタを共有する複数の生存区間が存在する場合には，生存区間のうち最小の始点と，最大の終点をそれぞれ始点と終点とする生存区間として扱うことにする．この定義により，述語付き Spiral Graph は従来の Spiral Graph と同様に扱える．

筆者らは，述語付き Spiral Graph に対して，述語が異なる生存区間におけるレジスタの共有にいくつかの方針を提案し，実験によって検証している[11]．

4. 必要レジスタ数の下界

Spiral Graph の幅 w_i とは，命令ステップ番号 i において同時に存在する生存区間の数である．あるグラ

☆ 述語付き Spiral Graph における副トラックは，Enhanced Modulo Scheduling 向けに拡張された Spiral Graph におけるサブトラック[7] とは別の概念であることに注意する．

フにおける幅の最大値を $W_{\max}$ という．プログラムで必要とするレジスタ数の下界は $W_{\max}$ となる．

述語付き Spiral Graph に対して同様に $W_{\max}$ を考える．生存区間が定義される述語の値が異なる場合には，これらの生存区間を同じ実レジスタに割り付けることができることから，$W_{\max}$ をレジスタ数の下界として利用するためには述語を考慮した $W_{\max}$ の定義が必要となる．

述語付き Spiral Graph において $W_{\max}$ をレジスタ数の下界として用いるためには，命令ステップ番号ごとに，共有可能なレジスタがすべて共有された場合の生存区間の数を求め，このグラフの幅の最大値を用いればよい．

定理 4.1（レジスタ共有条件） 互いに実レジスタを共有可能な生存区間は次の条件をともに満足する[10]．

- 同じパイプライン・ステージに存在する生存区間である．
- 生存区間が存在する場合の述語の値が異なる．■

証明 述語付き命令によって構成されたソフトウェア・パイプラインの実行モデルを考えると，異なるパイプライン・ステージに存在する生存区間どうしは，述語が異なっていても同時に生存する可能性がある．あるパイプライン・ステージの述語が真であっても偽であっても，同時に実行されている他のパイプライン・ステージの述語は真と偽のどちらの値もとりうるからである．

これに対して，述語が異なる同じパイプライン・ステージが同時に実行されることはありえないから，同じパイプライン・ステージにおける異なる述語が付された生存区間は同じ実レジスタを共有できる．■

たとえばサンプルプログラムにおいて $W_{\max}$ を考えると，命令ステップ番号 0 では，a_1，b_2，d_3，d_4，f_5，g_6，h_4，i_5 が存在している．生存区間に付した数字はパイプライン・ステージ番号である．このうち，d_4 と h_4，f_5 と i_5 は同じステージにおいて異なる述語の命令によって用いられているため，同じ実レジスタを共有することが可能である．よって w_0 は 6 となる．同様にして，$w_1 = 7$，$w_2 = 7$，$w_3 = 6$，$w_4 = 6$，$w_5 = 6$，$w_6 = 6$ であり，このグラフの $W_{\max}$ は 7 となる．

5. 割付けアルゴリズム

4 章で定義した必要レジスタ数の下界である $W_{\max}$ に対して，必要レジスタ数が $W_{\max}$ あるいは $W_{\max}+1$ となる多項式時間割付けアルゴリズムについて述べる．アルゴリズムの概要は以下のとおりである．

(1) グラフに埋め草を導入し，閉包とする．
(2) 述語付き命令による生存区間を組み合わせる昇格を行い，述語の付かない生存区間とする．
(3) 生存区間を連結してスライドカバーを作成する．
(4) スライドカバーどうしの埋め込みをする．

5.1 埋め草の導入

Spiral Graph において，生存区間の集合に対して，$0 \le t < II$（II: Initiation Interval）である長さ 1 の生存区間 $[t, t+1)$ を適当に加え，グラフの任意の命令ステップ番号におけるグラフの幅 w_i を $W_{\max}$ と同一にする．このような生存区間を埋め草[12] あるいは fictitious interval[6] と呼び，埋め草を加えたグラフを，もとのグラフの閉包と呼ぶ．埋め草は実際の生存区間ではないため，レジスタ割付け処理が終了したのちに除去する．

述語付き Spiral Graph においては，副トラックに存在する生存区間の幅を一定にするために，述語の付いた埋め草を用いる．述語の付いた埋め草を導入するためには，ある生存区間に対して同じ実レジスタを共有できる生存区間がない場合に，それとは異なる述語の付いた長さ 1 の生存区間を加える．なお，これによって，$W_{\max}$ が増えることはない．

複数の生存区間によって実レジスタを共有する場合には，述語の値が異なるだけでなく，生存区間の存在するステージ番号が同一であることも求められる．これを**図 5** に示すように，述語付き Spiral Graph をパイプライン・ステージの数だけ展開したグラフを用いて説明する．図 5 におけるらせんは，上から（主）トラック，述語が真の場合の副トラック，述語が偽の場合の副トラックを表す．このグラフにおいて，横軸の数字はパイプライン・ステージ番号，横軸の目盛は命令ステップ番号である．

条件判定命令，すなわち述語の定義命令は 2 ステージ目の後尾に存在するから，述語の 3 ステージ目の開始から生存区間 h の始点まで，さらに i の終点から g の終点までの区間に，長さ 1 の埋め草が追加される．

述語付き Spiral Graph に対しては，アルゴリズム 5.1 によって生成された埋め草を含むグラフを閉包と呼ぶことにする．なお，述語付きの生存区間が存在しない場合には，従来 Spiral Graph に埋め草を導入した場合と同様の結果を与える．

アルゴリズム 5.1（埋め草の導入）

(1) 述語付き命令により定義，参照される生存区間のみについて，命令ステップ番号ごとに述語付き命令による生存区間の数を調べる．ある生存区間に対して，レジスタ共有条件（定理 4.1）

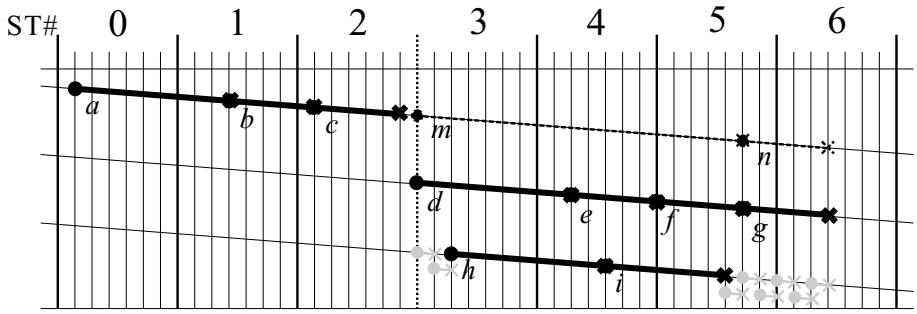

図 5 展開された述語付き Spiral Graph
Fig. 5 Expansion of Predicated Spiral Graph.

に従って実レジスタを共有できる生存区間が存在しない場合，これと実レジスタを共有可能な述語付きの埋め草を加える．

(2) 述語付き Spiral Graph 全体について，命令ステップ番号ごとにグラフの幅を調べ，幅が $W_{\max}$ と等しくなるように埋め草を加える． ■

5.2 述語の付いた生存区間の昇格

述語付き命令による生存区間をそうでない生存区間と同様に扱うために，述語付き Spiral Graph の閉包から，実レジスタを共有できる生存区間どうしを組み合わせる．述語付き生存区間に対する埋め草を導入したことによって，必ず共有可能である生存区間を発見することができる．これによって，最小の始点あるいは最大の終点においても実レジスタを共有可能な述語付き生存区間が必ず存在することから，3 章で述べた複数の生存区間の最小の始点と最大の終点を通常の生存区間と見なす方法とは異なり無駄な隙間が生じることはない．

アルゴリズム 5.2（昇格）

(1) 述語付き命令による生存区間がなければ終了する．

(2) 異なる述語を持つ生存区間どうしで，始点が同一である生存区間をすべてとりだす（図 5 における d の始点，あるいは g の始点がこれに相当する）．

(3) すべての生存区間の終点が揃っていた場合には，これまでにとりだした生存区間を隙間なく並べたものを，述語の付かない生存区間と見なすことにして，(1) にすすむ．

(4) 終点が揃っていなかった場合には，生存区間から最も終点が小さいものを選び，この終点と同一の述語で，同一の始点を持つ生存区間を隙間なく並べ，(3) にすすむ． ■

アルゴリズムの手順に対して以下の補題を証明する．

補題 5.1（始点の存在） アルゴリズム 5.2 の (2) において，始点が同一で，述語が異なる生存区間が必ず存在する． ■

証明 述語付き命令による生存区間は，述語の定義命令より後に現れるから，最も小さい始点が必ず存在する．最も小さい始点と，述語が異なり同一の始点を持つ生存区間が存在しない場合には，閉包を作る際に埋め草が導入される． ■

補題 5.2（終点の存在） アルゴリズム 5.2 の (3) において，終点が同一で，述語が異なる生存区間が必ず存在する． ■

証明 最も大きい終点と，述語が異なり同一の終点を持つ生存区間が存在しない場合には，閉包を作る際に埋め草が導入される． ■

補題 5.3（終点と始点の同一性） アルゴリズム 5.2 の (4) において，終点と同一の述語で同一の始点を持つ生存区間が必ず存在する． ■

証明 もしこの生存区間が存在しないとすると，異なる述語による生存区間と実レジスタを共有する生存区間が存在しないことになる．これは閉包の定義に矛盾する． ■

図 5 では，埋め草を含めたすべての述語付き生存区間を，$m = (\{d, e, f\}, \{\_, \_, h, i, \_\})$ と $n = (\{g\}, \{\_, \_, \_, \_, \_\})$ の 2 つの述語付き命令によらない生存区間として再生成できる．ここで，$\_$ は適当な埋め草を表す．

5.3 連結とスライド カバー

閉包を作成し昇格を行ったことで，$W_{\max}$ を増やすことなく，グラフの生存区間をすべて述語によらないそれらに変換することができた．このグラフに対して従来の Spiral Graph に対する多項式時間でのレジスタ数を最小化するレジスタ割付け方法を適用する．ここでは，展開していない述語付き Spiral Graph について考える．

定義 5.1（連結） 2 つの生存区間 v_i と v_j において，これらの隙間が 0，すなわち v_i の終了命令ステッ

プ番号と v_j の開始ステップ命令番号が同じであるとき，v_i と v_j はこの順序で連結可能であるという．■

ここでの連結は，命令の始点および終点ではなく，開始命令ステップ番号と終了命令ステップ番号であることに注意する．図 5 においては，a と c の始点は異なっているが，スケジュールされた命令では同じ命令スロットで定義されていることが図 3 から分かる．

定義 5.2（スライド カバー） 連結可能な生存区間を次々と連結し，連結された生存区間の開始命令ステップ番号と終了命令ステップ番号が等しい場合，これをスライド カバーと呼ぶ．■

定理 5.1（閉包の性質） Spiral Graph において，もとのグラフの閉包に含まれる生存区間はいくつかのスライド カバーに余すところなく作り分けることができる[7]．■

証明 閉包では，すべての命令ステップにおいて $W_{\max}$ が等しい．ある生存区間を連結するためにグラフから取り除くと，その生存区間の終了命令ステップ番号に等しい開始命令ステップ番号を持つ生存区間が必ず存在することになる．■

あるスライド カバーがすべての命令ステップに連結部分を持っているとは限らないため，スライド カバーは複数になる可能性がある．

定理 5.2（$W_{\max}+1$ 割付け） 生成されたスライド カバーの開始命令ステップ番号を単調増加となる順序で並べることで，$W_{\max}+1$ 本のレジスタに収めるレジスタ割付けが行える[7]．■

サンプルプログラムに対してスライド カバーを生成し，$\{d,\ldots\}$，$\{a,\ldots\}$ のスライド カバーの順に並べたものが**図 6** である．スライド カバーは開始命令ステップ番号と終了命令ステップ番号が同一であるから，単調増加となる順序で並べることで，スライド カバー $\{d,\ldots\}$ の終点と同一の実レジスタに，スライド カバー $\{a,\ldots\}$ の始点を割り付けることができる．これにより，$W_{\max}+1=8$ 本のレジスタに割り付けられていることが分かる．

5.4 最適割付けのための条件

必要レジスタ数が $W_{\max}$ で割り付けられるならば，割付け結果のレジスタ数はただちに最小であるといえる．

レジスタ改名命令にまたがって存在する生存区間は，レジスタ改名前に占有していたレジスタに，改名後に占有していたそれを加え，通常よりも 1 つ多いレジスタを使用することになる．そこで，スライド カバーがレジスタ改名命令をできる限りまたがないように構成する必要がある．そのため，埋め草を含めたグラフの

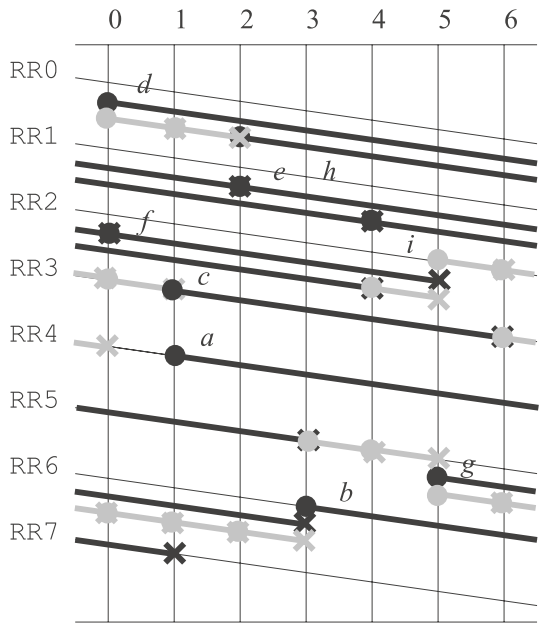

図 6 $W_{\max}+1$ での割付け例
Fig. 6 Result of allocation with $W_{\max}+1$.

閉包において，すべての命令ステップ番号におけるグラフの幅が $W_{\max}$ に等しいとき，このグラフを $W_{\max}$ 本のレジスタに割り付けるためには，以下の条件を満足する必要がある．

定理 5.3（最適割付けの条件） Spiral Graph においてレジスタ割付け結果が $W_{\max}$ と一致するのは，次の条件がともに満たされたときである．

(1) グラフ全体でスライド カバーが 1 つになること．
(2) レジスタ改名命令直前の生存区間が埋め草であること．■

このようなスライド カバーを構成しようとするアルゴリズムが存在する[7]．グラフ全体でスライド カバーを 1 つにするために以下の補題を用いる．

補題 5.4（スライド カバーの組み換え） スライド カバーは，これに含まれる任意の生存区間の開始命令ステップ番号を始点とするスライド カバーに組み換えることができる．■

証明 スライド カバーは生存区間を連結したものであるから，互いに同一の開始ステップ番号と終了ステップ番号を持つ生存区間が存在する．さらに，もとのスライド カバーの始点と終点が同一の命令ステップ番号であるから，生存区間の先頭を始点として生存区間を連結することができる．■

補題 5.5（スライド カバーの埋め込み） 2 つのスライド カバーに，同一の開始命令ステップ番号を持つ生存区間が含まれているとき，片方のスライド カ

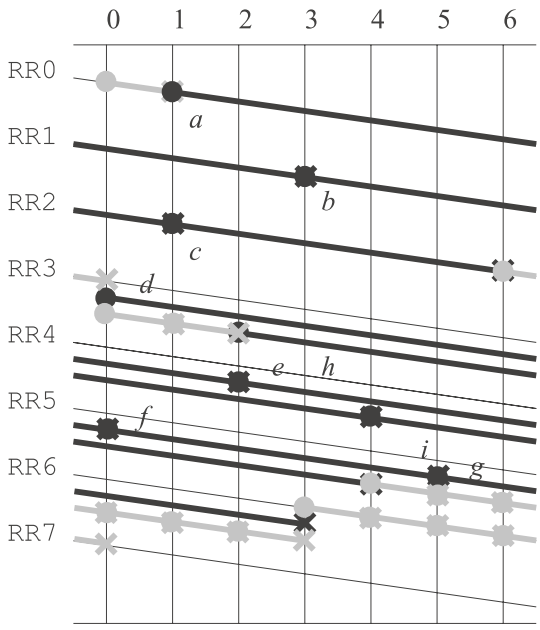

図 7 $W_{\max}$ での割付け例
Fig. 7 Result of allocation with $W_{\max}$.

バーを他方に埋め込み，1 つのスライド カバーとして再構成できる． ■

証明 2 つのスライド カバーを組み換えて，同一の開始命令ステップ番号を持つ生存区間から始まるスライド カバーとする．これらは 1 つのスライド カバーとして連結できる．再度もとの開始命令ステップ番号から始まるスライド カバーとして組み換えることができる． ■

図 7 は，$\{\_, c\}$ に $\{a, \_, \_, (\{g\}, \{\_, \ldots\}), b\}$ を埋め込み，全体のスライド カバーの最後，すなわちレジスタ改名命令直前が埋め草となるように組み換えた例である．この場合，レジスタ改名命令である`br.ctop`をはさんだ，`RR6` と `RR7` にまたがる生存区間は埋め草であるため割付け結果からは除かれ，グラフの`RR0` から`RR6`，$W_{\max} = 7$ 本でのレジスタ割付けが可能であった．

5.5 多項式時間最適割付けの計算量の試算

それぞれの手順について，計算量の試算を行う．グラフ中に存在する生存区間の総数を N，うち述語付き命令による生存区間の数を N_p とする．II は繰返し開始間隔（Initiation Interval）である．

(1) **述語付きの生存区間に対する埋め草の導入**
展開した Spiral Graph で考える．$N - N_p$ 個の生存区間について，配列で表現した命令スロットごとの幅を調べる必要があるので $\mathrm{O}((N - N_p) \cdot (s \cdot II))$（ただし s は述語付きのパイプラインステージ数）．
さらに，どの述語の生存区間が不足しているか調べるために，条件分岐の数を C_n としたとき，$\mathrm{O}(2^{C_n} \cdot (s \cdot II))$

(2) **幅を $W_{\max}$ に揃える埋め草の導入**[7]
$\mathrm{O}(N^2)$

(3) **昇格**
最大 N_p 個の生存区間に対して，すべての場合で終点が揃っているかを調べるために，条件分岐の数を C_n として $\mathrm{O}(2^{C_n} \cdot N_p)$

(4) **スライド カバーの生成，組替え，埋め込み**[7]
$\mathrm{O}(N^2)$

生存区間は命令によって定義あるいは参照される．N 個の生存区間が生成され使用されるためには 2 つ以上の命令を必要とするので，II（Initiation Interval）はたかだか $2N$ である．

よって，条件分岐の数が 1 つである場合には，必要レジスタ数が $W_{\max}$ あるいは $W_{\max} + 1$ とするレジスタ割付けは $\mathrm{O}(N^2)$ の多項式時間で行えることが保証される．

条件分岐が複数の場合には，最大で $\mathrm{O}(2^{C_n} \cdot N)$ の計算量が必要となる．条件分岐の数が多いときにはつねに述語付き命令実行機構を使うことも可能だが，その場合 1 回のループで実行される命令の充填率が下がり，性能が低下する．そのため，特に生存区間数 (N) や命令数の多いループにおいては，命令スケジューラは通常の条件判定およびジャンプ命令を構成することになる．よって，2^{C_n} が N に比べて非常に大きくなる可能性は少なく，計算量は $\mathrm{O}(N^2)$ であることが強く示唆される．

6. 関 連 研 究

述語付き命令実行機構向けのレジスタ割付け方法として，干渉グラフを用いた方法が提案されている[5]．この方法では，生存区間を表現するノードに付帯した述語が互い異なる場合には，これらを束ねた（bundle）干渉グラフを再生成してから割付けする方法である．

この方法は，ソフトウェア・パイプラインやレジスタ改名機構への対応が難しいこともあるが，束ねる条件がヒューリスティックであるため，割付けにより得られたレジスタ割付けが最適である保証がない．これに対して本論文におけるレジスタ割付け方法は，レジスタ割付け結果のレジスタ数における最適性について述べている．

レジスタ改名機構が存在しない場合のレジスタ割付け方法として，meeting graph[6] による方法が提案さ

れている．この方法はプログラム中の生存区間の連続性を用いてレジスタ割付けする方法であり，Spiral Graph における方法[7] とはまったく別に fictitious interval や回路を用いた，レジスタ数を最小とする多項式時間割付けアルゴリズムを提案している．

この方法においても，レジスタ改名機構が存在する場合には，必要レジスタ数を $W_{\max}$ 以上かつ $W_{\max}+1$ 以下に収めることができると述べられている．これに対して本論文におけるレジスタ割付け方法は，述語付き命令実行機構を考慮し，生存区間に述語が付いたより複雑な問題を対象としていること，および必要レジスタ数が $W_{\max}$ に収まる条件を提示している点が異なる．

7. まとめと今後の課題

本論文では，インテル IA-64 アーキテクチャに代表される，レジスタ改名機構と述語付き命令実行機構を備えたアーキテクチャに対して，Spiral Graph を用いたレジスタ割付け方法を提案した．

まず，レジスタ改名機構向けの Spiral Graph を述語付き命令実行機構向けに拡張した，述語付き Spiral Graph を提案した．

次に，述語付き Spiral Graph に対してグラフの幅である $W_{\max}$ を定義した．$W_{\max}$ は必要レジスタ数の下界として用いることができる．

さらに，必要レジスタ数を $W_{\max}$ 以上 $W_{\max}+1$ 以下に収めることのできる，必要レジスタ数における最適割付けアルゴリズムを提案した．このアルゴリズムは $O(N^2)$ の多項式時間での割付けが可能である．

将来的には，実際の最適化コンパイラのレジスタ割付け器として実装を行い，標準的なベンチマークや実際のプログラムにおいて性能を調査することがあげられる．

参 考 文 献

1) Allan, V.H., Jones, R.B., Lee, R.M. and Allan, S.J.: Software Pipelining, *ACM Computing Surveys*, Vol.27, No.3, pp.367–432 (1995).
2) Allen, J.R., Kennedy, K., Porterfield, C. and Warren, J.: Conversion of Control Dependence to Deta Dependence, *Proc. 10th ACM Symposium on Principles of Programming Languages* (*POPL*), pp.177–188 (1983).
3) Dehnert, J.C., Hsu, P.Y.-T. and Bratt, J.P.: Overlapped Loop Support in the Cydra 5, *Proc. 3rd International Conference on Architectural Support for Programming Languages and Operating Systems*, pp.26–38 (1989).
4) Duling, C., Krishnaiyer, R., Kulkarni, D., Lavery, D., Li, W., NG, J. and Sehr, D.: An Overview of the Intel IA-64 Compiler, *Technology Journal*, Intel (1999).
5) Eichenberger, A.E. and Davidson, E.S.: Register Allocation for Predicated Code, *Proc. 28th Annual International Symposium on Microarchitecture* (*MICRO-28*), pp.180–191 (1995).
6) Eisenbeis, C., Lelait, S. and Marmol, B.: The meeting graph : A new model for loop cyclic register allocation, *Proc. 5th Workshop on Compilers for Parallel Computers* (*CPC95*), pp.503–516 (1995).
7) 秡川友宏：スライドウィンドウを考慮したレジスタ割付の研究，博士課程工学研究科博士論文，筑波大学 (2000).
8) 秡川友宏，添野元秀，山下義行，中田育男：スライドウィンドウを考慮したレジスタ割付，情報処理学会論文誌，Vol.39, No.9, pp.2684–2694 (1998).
9) Intel: *Intel Itanium Architecture Software Developer's Manual*, Vol.1, Vol.2, Vol.3, Vol.4 rev.1.1 (2000).
10) 糸賀裕弥，秡川友宏，山下義行，中田育男：条件分岐を考慮したソフトウェアパイプラインにおけるレジスタ割付け，電子情報通信学会論文誌 DI, Vol.J85-D-I, No.1, pp.31–39 (2002).
11) Itoga, H., Haraikawa, T., Yamashita, Y. and Tanaka, J.: Register Allocation for Software Pipelining with Predication using Spiral Graph, *Proc. International Symposium on Future Software Technology* (*ISFST2001*), pp.58–65 (2001).
12) 中田育男：コンパイラの構成と最適化，朝倉書店 (1999).

(平成 14 年 2 月 20 日受付)
(平成 14 年 5 月 16 日採録)

糸賀 裕弥（正会員）

1972 年生．1995 年筑波大学第 3 学群工学システム学類中途退学．2002 年同大学大学院博士課程工学研究科修了．工学博士．2001 年 7 月から茨城県工業技術センター流動研究員．2002 年 4 月から筑波大学先端学際領域研究センター（TARA）マルチメディア情報研究アスペクト「実世界指向インタラクションの研究」客員研究員．並列化，最適化コンパイラの研究に従事．モバイルデバイスのシステムソフトウェア，ヒューマンインタフェース，ネットワークセキュリティにも興味を持っている．ACM，電子情報通信学会，日本ソフトウェア科学会各会員．

山下 義行（正会員）

1959 年生．1982 年大阪大学理学部物理学科卒業，日立マイクロコンピュータ・エンジニアリング（株）入社．1986 年退社．1987 年筑波大学大学院博士課程工学研究科電子・情報工学専攻入学．1989 年退学，東京大学大型計算機センター助手．1992 年筑波大学電子・情報工学系講師．1995 年〜2001 年同助教授．2001 年〜佐賀大学理工学部教授．工学博士．プログラミング言語，コンピュータグラフィクスの研究に従事，日本ソフトウェア科学会会員．

田中 二郎（正会員）

1975 年東京大学理学部卒業．1977 年同大学大学院修士課程修了．1984 年米国ユタ大学計算機科学科博士課程修了，Ph.D. in Computer Science．現在，筑波大学電子・情報工学系教授．プログラミング一般やヒューマンインタフェースに関する研究を行っている．最近は新しいプログラミング言語の枠組みとして，スクリプト言語に興味を持っている．2001 年 4 月から筑波大学先端学際領域研究センター（TARA）マルチメディア情報研究アスペクトで「実世界指向インタラクションの研究」の研究代表者をつとめている．2002 年 4 月から筑波大学第 3 学群情報学類長．ACM，IEEE Computer Society，電子情報通信学会，人工知能学会，日本ソフトウェア科学会各会員．