4R-04

複数パスのソフトウェア・パイプラインにおけるレジスタ割付

糸賀 裕弥（筑波大学）† 山下 義行（筑波大学）††

1. はじめに

コンパイラで行なわれるレジスタ割付処理においては，干渉グラフと，グラフに対する k-彩色法を基本とした方法で一般的には十分である．

しかし，条件分岐を含むループをソフトウェア・パイプライン化した場合の干渉グラフに対しては，単なる k-彩色法では不十分な場合があることが判明した．

この場合は，干渉グラフに対して k-彩色法を適用したのちに，さらに性質の異なる彩色法を，あわせて適用することでより良好なレジスタ割付結果が得られることがわかった．

2. レジスタ割付における k-彩色法

一般的なレジスタ割付においては，まず，レジスタにあたる一時変数の生存区間を頂点とし，同時に生存する頂点どうしを枝で結んだ干渉グラフを生成する．次に，彩色アルゴリズムを適用して，枝で結ばれた頂点どうしを別の色に彩色し，それぞれの色を別の実レジスタとみなすことでレジスタ割付を行なう．

2.1 k 彩色法の統計的性質

k-彩色法[1] は，干渉グラフに対する効率の良い彩色法の1つである．これは干渉グラフが k 色以下で彩色可能かを，干渉グラフの大きさに対して線型時間で判定できる．可能であった場合には，同時にグラフの彩色結果も得ることができる[2]．

与えられた干渉グラフと頂点数が同等である完全グラフの総枝数に対して，もとのグラフの枝数を，枝の存在確率と呼ぶ．グラフ理論で扱う彩色問題では，頂点数が 1000，枝の存在確率が 0.5 程度の大きなグラフを想定しており，これに対する種々の近似彩色アルゴリズムによる彩色数は $130-95$ 程度になる[3]．

このようなグラフに単純な k-彩色法を適用した場合，彩色の期待値は頂点数 N と存在確率 p の積である $N \times p$ に近づき，彩色数の期待値は 500 程度になる．つまり，k-彩色法が有効であるのは，グラフが比較的小さく，枝が少ない，一般的なコンパイラにおけるレジスタ割付のような問題に限られることがわかる．

2.2 条件分岐を含む場合のソフトウェア・パイプラインへの適用

条件分岐を含むループのソフトウェア・パイプライン[4] コードに対して，干渉グラフと k-彩色法を用いたレジスタ割付を行なう場合を考える．紙面の都合上詳細は述べないが，次の手順でコードを生成する．

(1) 述語（predicate）付き命令をソフトウェアパイプライン化してカーネル部を生成する．

(2) 通常のアーキテクチャで述語付き命令を実行可能とするために，カーネルを複数に展開する．

これに対するレジスタ割付としては，展開前のカーネル部に対してあらかじめレジスタ割付を行なう方法と，展開後のコードに対してレジスタ割付を行なう方法が考えられる．この2つのレジスタ割付方法の比較において，k-彩色法の彩色結果が他の彩色法に比べて良好な結果を与えていないことが判明している[5]．

しかし，展開後のコードと，展開前のコードは本質的に同じグラフであり，双方の干渉グラフにおける実際の彩色数は同じになるはずである☆．

展開後のコードに対する干渉グラフは，通常のコードのものと変わらないため k-彩色法でも十分である．しかし，展開前のカーネル部に対する干渉グラフでは，実行時に別のコード上に存在する頂点と干渉枝が小さなグラフ上に表現されるため，頂点数の減少に比べて枝の数が減少していない．つまり，k-彩色法の統計的性質で述べたように，枝の存在確率が大きく，彩色数が大きく判定されてしまっていたことになる．

例えばリスト1のサンプルプログラムから，コードの展開前に生成した干渉グラフは図2のようになる．頂点の度数（頂点に付した数字）から，k-彩色法では3色では彩色不可能であると判定される．しかし実際には，あらかじめグラフに彩色されているように，3

Register Allocation method for Multi-Pass Software Pipelining.

† Hiroya ITOGA, Doctoral Program in Engineering, University of Tsukuba.

†† Yoshiyuki YAMASHITA, Institute of Engineering Mechanics and Systems, University of Tsukuba.

☆ 展開前のカーネル部から生成した干渉グラフは，非常にまれであるが，展開後のコードの一部分では干渉しない頂点どうしを，常に干渉すると表現することがある．全解探索に相当する彩色実験により，この差は非常に小さいことが判明している．

```
IF (( C(I)+1.0 )**2.0 .EQ. 1.0 ) THEN
  Z(I) = X(I) + Y(I)
ELSE
  IF (( C(I)+1.0 )**2.0 .GT. 1.0 ) THEN
    Z(I) = W(I)
  ELSE
    Z(I) = V(I)
  END IF
END IF
```

図 1 サンプルプログラム
Fig. 1 A sample program.

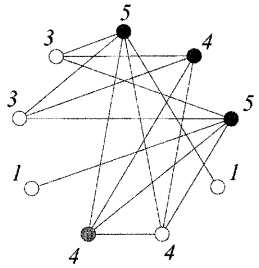

図 2 干渉グラフの例
Fig. 2 Interference graph of the sample program.

色で彩色が可能である.

3. k-彩色法の改良

k-彩色法は，グラフ彩色における比較的単純な近似アルゴリズムの 1 つである頂点の度数の小さいものを最後に彩色する (SMALLEST LAST) 手法の応用であると考えられる.

同様に比較的単純な近似アルゴリズムとして，度数の大きいものを先に彩色する (LARGEST FIRST) 方法がある[3].

本来 k 色で彩色可能なグラフが，k-彩色法において彩色不可能と判定されるのは，グラフの枝の存在確率が大きい場合，すなわち，k-彩色法で想定する以上に枝の数と頂点ごとの度数が大きこいことが考えられる.

そこで，k-彩色法を適用して彩色不能として残った干渉グラフに対して度数の大きいものを先に彩色するアルゴリズムを適用する．これによって実際に k 色での彩色が可能であれば，もとの干渉グラフも k 色で彩色可能となる.

4. 実　　験

以上を確認するために，実験を行なった.

乱数を用いてレジスタに相当する一時変数を 1000 例生成し，条件分岐を含むループのソフトウェアパイプライン法にもとづいてスケジュールし，干渉グラフを生成した.

これに対して，色数を増やしながら k-彩色法を適用した．さらに，彩色できなかったと判定された場合の残存グラフを，頂点の度数の大きいものから彩色するという簡単な彩色法を用いて，k 色以下で彩色可能かを調査した.

通常のコードに相当するカーネル展開後のコードの場合には，残存グラフの彩色によって 1000 例中わずかに 3 例において，彩色の改善がみられるのみだった．しかし，展開前のカーネルに対するレジスタ割付においては同じ 1000 例に対して 494 例に彩色の改善がみられた．なお，それぞれの干渉グラフにおけるの枝の存在確率は 0.04 と 0.15 であった.

以上のように，展開前のカーネルに対する干渉グラフは，枝の存在確率が大きく，k-彩色法が適用しにくい性質を持つことが確認された.

5. まとめと今後の課題

条件分岐を含むループのソフトウェア・パイプラインに対するレジスタ割付においては，一般的なコンパイラで有効とされる k-彩色法によるレジスタ割付方法では十分でないことが判明した．これは，枝の存在確率が通常のコードよりも大きいことが理由として考えられる.

このとき，k 彩色法に加えて，性質の異なる彩色法をあわせて適用することで，さらに良好な彩色結果が得られることがわかった.

今後の課題としては，k-彩色法で彩色不能と判定された場合の仮のスピル処理として，度数が k 以上のノードの除去行ない k-彩色を続けるアルゴリズム[6] との比較がある．また，今後主流になるであろう述語付き命令をハードウェアで処理可能なプロセッサに対する，本アルゴリズムの有効性を探ることがあげられる.

参 考 文 献

1) 佐々政孝: プログラミング言語処理系, 岩波書店 (1989).
2) Chatin, G. J., Auslander, M. A., Chandra, A. K., Cocke, J., Hopkins, M. E. and Markstein, P. W.: Register Allocation via Coloring, *Computer Languages*, Vol. 6, pp. 47–57 (1981).
3) Manvel, B.: Extremely Greedy Coloring Alogorithms, *Proc. of the 1st Colorado Symp. on Graph Theory*, pp. 257–270 (1985).
4) 山下義行, 中田育男: ループ中に条件分岐を含む場合の最適なソフトウェア・パイプライニング, *JSPP'94*, pp. 17–24 (1994).
5) 糸賀裕弥, 秋川友宏, 山下義行, 中田育男: 条件分岐を考慮したソフトウェア・パイプライニングにおけるレジスタ割付, *JSPP'99*, pp. 39–46 (1999).
6) Appel, A. W.: *modern compiler implementation in Java*, Cambridge University Press (1998).