

レジスタ割り付けにおける Optimistic Register Coalescing の 実装と評価

Implementation and Evaluation of a Optimistic Register Coalescing
in the Register Allocation

副島 佑介[†] 佐々 政孝[†]
Yusuke Soejima Masataka Sassa

[†] 東京工業大学 情報理工学研究科 数理・計算科学専攻
Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology
soejima.y.aa@m.titech.ac.jp

プログラム中では、レジスタを用いた命令はメモリを参照する命令よりも一般に速い。そこで効率の良いコードをコンパイラにおいて生成するには、変数等を出来るだけレジスタに割り当て、レジスタを有効に活用することが大切である。本研究では適用対象である COINS [1] について、新しいレジスタ割当ての実装を行った。COINS とはコンパイラ研究の基盤となる共通のコンパイラの作成を目標として研究が進められているコンパイラ・インフラストラクチャである。従来実装されていた COINS 上のレジスタ割り付けアルゴリズムは *iterated coalescing* [2] と呼ばれるものをベースにしていたが、今回は *optimistic coalescing* [3] と呼ばれるアルゴリズムを COINS 上に実装し、評価した。

1 はじめに

1.1 レジスタ割り付け

レジスタ割り付け (Register Allocation) (レジスタ割り当てとも呼ぶ) とは、プログラム内の多数の変数を少数の CPU レジスタに多重化する技法で、その目標は、プログラムの実行速度を最小化すべく多くのオペランドをレジスタに保持するようにする事である。

1.2 研究概要

現在、COINS [1] で用いられているレジスタ割り付けアルゴリズムは、グラフ彩色アルゴリズムの改良型である George-Appel の「*iterated register coalescing*」 [2] をベースにしている。この方法は *coalescing* (合併) を「保守的」にしか行わない。今回はそのアルゴリズムを Park-Moon の「*optimistic register coalescing*」 [3] をベースにしたアルゴリズムに変えて COINS 上に実装を行った。この方法は *coalescing* を「楽観的」に行い、*iterated register coalescing* よりもよいレジスタ割り付けができること主張されている。

1.3 COINS

COINS [1] とは、コンパイラ研究の基盤となる共通のコンパイラの作成を目的として開発された、並列化コンパイラ向け共通インフラストラクチャである。COINS のバックエンドはフロントエンドが出力した中間コードファイル (LIR) を読み込んで機械語を生成する。今回実装した部分はこのバックエンドのレジスタ割り付け部分にあたる。

2 グラフ彩色アルゴリズム

本研究のレジスタ割り付けは干渉グラフを用いて行う。以下は簡単な干渉グラフの説明とグラフ彩色アルゴリズムの説明である。

2.1 干渉グラフによるレジスタ割り付け

まず各変数が生きている区間を生存区間という。これを元に変数の生存区間グラフを作成し、そしてこの情報を元に各変数をノードとして、生存区間が互いに重なっている変数同士を互いにグラフ上で辺 (無向辺) で結んだものを干渉グラフという。この干渉グラフについて、使用できるレジスタを色に見立て、辺で結ばれたノード同士が同じ色にならないよう各ノードに色を塗り分けていく事をグラフ彩色という。簡単な例を以下に示す。

```

do i = ...
  1: a = c * d
  2: b = d + a
  3: c = a - 5
  4: d = b * a
end do

```

このプログラムから得られる生存区間についてそれを干渉グラフで表現したものが図 1 の (a) となる。

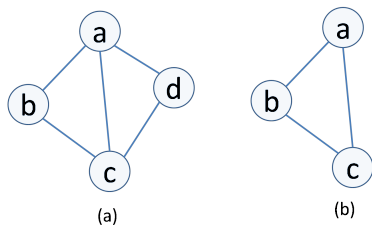


図 1: 生存区間の干渉グラフの例

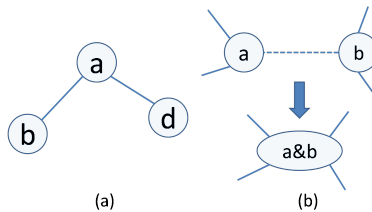


図 2: 生存区間の干渉グラフ (スピルと合併の例)

この例の場合、仮に使えるレジスタ数を 3 として (それぞれ R1, R2, R3 とする) 割り付けを考える。まずグラフの単純化 (simplify) [4] を行う。単純化とは、グラフからノードを省きスタックに積んでいくことであり、その対象となるノードは必ず割り付けが可能であるノードである。例で説明をすると、上の図 1(a) では b と d はそれぞれ二つ (a と c) のノードとしか干渉していないのでレジスタ数が 3 であるこの場合、必ず割り付けられるレジスタが存在すると言える。そのようなノードをグラフから取り除いていき、彩色問題及びグラフ構造を単純にしていく事を単純化という。図 1 の (b) がノード d を単純化した後のグラフになっていて、(b) のグラフが彩色可能ならば、元の (a) のグラフも彩色可能と言える。単純化を終えると全ノードはスタックに積まれており、

スタックの上から適当なレジスタを選んで割り付けを行う。(表 1)

a	R1
b	R2
c	R3
d	R2
スタック	割り当て

表 1: スタックと割り当て

2.2 スピルと合併

ある変数について、割り付け可能なレジスタが足りなくなれば、最初に述べたようにメモリを参照し割り当てる事となる。これをスピル (spill) と言う。スピルが起こると、その変数 (ノード) を除いた干渉グラフについてグラフ彩色をまた考える事となる。グラフ上でのスピルの例として、図 1 の (a) について考える。もし使用できるレジスタが 2 つだった場合、このグラフは彩色が出来ない。そのような場合に、グラフ上からスピルするノードを選ぶ (この場合ノード c とする)。そしてグラフ上から省くと残りは図 2 の (a) のようになり、これはレジスタ数 2 で割り付けることが可能になる。

またプログラム中に

```
a = b
```

というようなコピー文 (代入文) が存在し、かつ互いにノードで繋がっていないければ (つまり生存区間に重なりがなければ) 同じレジスタを割り付ける事が可能である。これを合併 (coalesce) といい、干渉グラフ上では合併可能なノード同士を点線で結んで表す。

例としては図 2 の (b) を見られたい。ノード a, b が合併候補となり、干渉していない場合はこのようにノードをくっつけて合併を行う。(合併後のノードが a & b となる)

合併する事は最適化のコピー伝播と同等であり、レジスタの節約にも繋がる。つまり、よい割り付けアルゴリズムはより多くの合併を行い、かつより少ないスピルでレジスタ割り付けを行う。

2.3 基本彩色アルゴリズム

基本的な干渉グラフを用いた彩色アルゴリズムについて説明をする。

このグラフ彩色には最適に解く効率のよいアルゴリズムは存在せず、様々なヒューリスティックなアルゴリズムが考えられている。基本的なそのアルゴリズムとしては Chaitin の提案したもの [7] があげられる。アルゴリズムの流れとしては図 3(a) であり、グラフ上の合併候補を出来るだけ合併 (aggressive coalescing [7]) したのちに、単純化を行い、スピルとなるものを省いていく手法である。

これはレジスタが無限にあると仮定し中間表現を変換していき、機械語に直す前に、マシンごとに定まった有限のレジスタを、干渉グラフによる情報を元に彩色アルゴリズムで求めた彩色パターンで割り当てていくという手法である。

この aggressive coalescing を用いた彩色アルゴリズムに関しては多くの改良案が提案されており、今回扱う iterated register coalescing, optimistic register coalescing がそれに当たる。

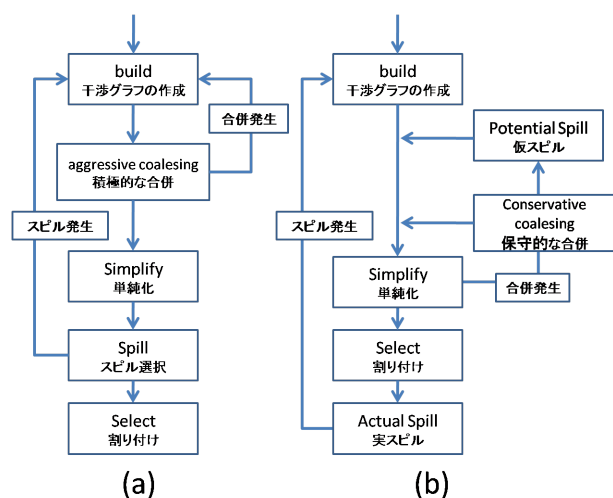


図 3: 各アルゴリズムのフェーズ

3 彩色アルゴリズム

3.1 従来 COINS 上で実装されてきたアルゴリズム

従来の COINS 上で実装されてきた彩色アルゴリズムについて説明をする。図 3 の (b) がアルゴリズムの流れである。この手法は、George-Appel の「iterated register coalescing」[2] をベースとしており、保守的な合併とスピルを二段階に分ける方法 [5] を組み合わせた彩色アルゴリズムである。

この手法は、割り付け可能性を損なうかもしれない合併候補はスピルを恐れ合併を行わず、安全な合

併と単純化、仮スピルを繰り返すことでグラフの状態を彩色可能性を損なわず簡略化していく。また保守的な合併により、合併後のノードがスピルの対象となる事を回避している。

3.2 問題点

従来の Iterated Register Coalescing には以下の問題点がある。

- 利点
保守的な合併により、合併ノードによるスピルを回避している。
- 問題点
保守的な合併の際、合併を恐れた合併候補は合併しても必ずスピルされる訳ではない。よって幾らかの合併は無駄に諦めてしまっている。また合併によりグラフの状態が良くなる可能性もあるが保守的な合併により見逃している。

これより改善したい点として

- 合併をさらに多く行う
- 合併ノードのスピルの数は増やしたくない

があげられる。これらを満たすアルゴリズムが次に紹介するアルゴリズムである。

3.3 今回 COINS 上で実装を行ったアルゴリズム

今回 COINS 上で実装を行った彩色アルゴリズムについて説明をする。図 4 の (a) がアルゴリズムの流れである。この手法は、Park-Moon の optimistic coalescing [3] をベースとしており、積極的な合併と、合併取り止めを用いたアルゴリズムである。

3.4 合併取り止め (Undo Coalescing)

合併取り止めとは、積極的に合併をし、かつ合併ノードのスピルを防ぐ為の手法である。合併ノードに対してスピルが必要になった場合にその合併を取り止め前の状態に戻すことで彩色を考え直すというアイデアである。具体的に図で説明をする。図 4(b) のグラフの彩色を考える。(使用可能レジスタは 2 つ) 積極的に合併候補を全て合併すると図 5(a) のようになる。この状態ではレジスタ 2 では割り当てが出来ず、合併ノードの中でどれかスピルの候補とならないといけな。 (図では b & f) しかし、ここで合併

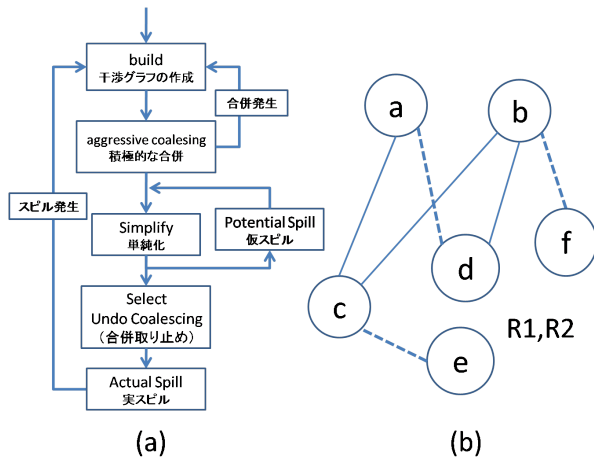


図 4: Optimistic Coalescing と干渉グラフ

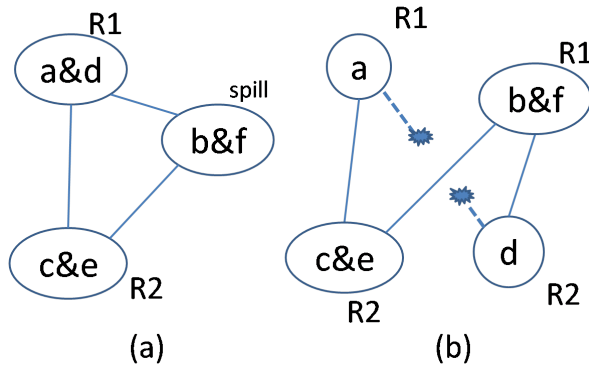


図 5: Undo Coalescing の例

取り止めで図 5(b) のようにする事で 2 つのレジスタで割り付ける事が可能となる。

4 実装について

実装には COINS のバージョン 1.4.4.1 を採用し、LIR(低水準中間表現) のバックエンドにおけるレジスタ割り付け (Register Allocation) 部分を変更した。今回用いた COINS のバックエンド上では手続きごとにレジスタ割り付けを行っており、本実装もそれと同様とする [8]。

5 実験結果と考察

本節では実装したレジスタ割り付けアルゴリズムの実験結果とそれに対する考察を行う。テストプログラムには、SPEC CPU2000 ベンチマークを用いた。比較するアルゴリズムは

- 従来の COINS のアルゴリズム (iterated coalescing)
- Chaitin のアルゴリズム (aggressive coalescing)
- 今回実装したアルゴリズム (optimistic coalescing)

とした。比較対象は

- 合併数・スピル数
- 目的コードの実行時間
- コンパイル時間

の 3 項目とした。

benchmark	iterated	aggressive	optimistic
164.zip	4	4	4
181.mcf	0	0	0
254.gap	27	28	27
256.bzip2	0	0	0
300.twolf	61	64	61
171.swim	3	3	3
172.mgrid	0	0	0
177.mesa	872	914	867
179.art	15	15	15
183.earthquake	23	23	23
188.ammp	613	619	613

表 2: スピル数

benchmark	iterated	aggressive	optimistic
164.zip	1353	1358	1358
181.mcf	249	248	248
254.gap	11575	11582	11582
256.bzip2	705	705	705
300.twolf	5843	5846	5849
171.swim	129	129	129
172.mgrid	266	266	266
177.mesa	7723	7819	7831
179.art	301	301	301
183.earthquake	562	562	562
188.ammp	2560	2559	2562

表 3: 合併数

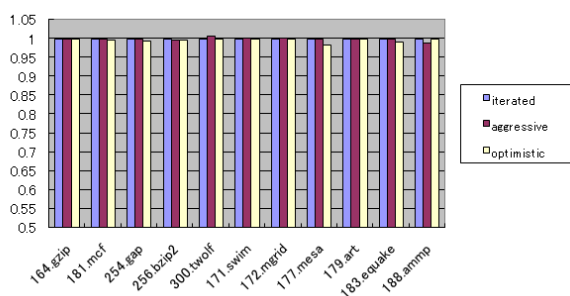


図 6: 目的コードの実行時間 (Iterated を 1 とした)

benchmark	iterated	aggressive	optimistic
164.gzip	49	50	49
181.mcf	33	33	32
254.gap	232	232	230
256.bzip2	14	14	14
300.twolf	287	277	274
171.swim	5	6	5
172.mgrid	8	8	8
177.mesa	386	378	379
179.art	9	7	7
183.earthquake	9	8	8
188.ammp	131	125	125

表 4: コンパイル時間 (単位: 秒)

スピル数・合併数については表 2 や表 3 のようになった。全体的に小さいプログラムでは差が出ていない。これはマシンのレジスタ数が多い事が要因である。大きめのプログラムに対しては Optimistic いくらか効果が見られた。

まずスピル数 (少ない方がよい) については、大きい差は見られなかった。Optimistic は iterated よりもスピル数が増える事がなく、これは aggressive の結果を見てわかるように合併取り止めによる効果が出たと言える。合併数 (一般的に多い方がよい) については、Optimistic はいくらか向上している。aggressive のアルゴリズムに比べても合併数が向上しているのは合併取り止めによったものである。特に 177.mesa のような大きいプログラムに対しては iterated に比べ、100 個ほどの差が見られた。

目的コードの実行時間についてはほとんどのプログラムでは時間の変化は見られなかった。(図 6) やはり合併数およびスピル数にそこまでの変化が見ら

れないことが原因と思われる。しかし 177.mesa のようにスピル数合併数ともに改善している物に対しては 2 % 程の改善が見られた。

コンパイル時間については幾らかの改善が見られた。(表 4) これはアルゴリズムの構造上、合併、単純化を繰り返す iterated のアルゴリズムよりも、まず全合併ノードを合併してしまう aggressive, optimistic のアルゴリズムの方が時間を取らない事による。300.twolf や 177.mesa などでは 10 秒近くの差が見られた。

6 まとめと今後の課題

総じてみると、大きめのプログラムに対しては、今回実装したアルゴリズムは従来の割り付け性能と同程度以上の効果を発揮し、かつコンパイル時間についても改善が見られた。

今回の手法は、扱う変数が多くなるような場合には効果が見られるが、そうでない場合には、大概の合併を諦めなくてよく、またスピルされる変数も固定されてしまい結果に差が表れなかった。

今後の課題として次の 2 つがあげられる。1 つ目としては、レジスタ数の少ないマシンでの性能の確認。2 つ目に、COINS では各手続きごとへのレジスタ割り付けしか行われていなく、スピルする変数については大域的な情報を一切無視しているので、これを改善した割り付け手法の実装が挙げられる。

謝辞

本研究の一部は科学研究費補助金の援助を受けた。

参考文献

- [1] COINS Project. Coins project home page. <http://www.coins-project.org/>.
- [2] George, L. and Appel, A. W Iterated register coalescing. ACM Transactions on Programming Languages and Systems 1996
- [3] Park, J. and Moon, S.M Optimistic register coalescing. ACM Transactions on Programming Languages and Systems 2004
- [4] 中田育男. コンパイラの構成と最適化. 朝倉書店.1999
- [5] Briggs, P. Register allocation via graph coloring. Ph.D. dissertation, Rice University 1992
- [6] Briggs, P.,Cooper, K. D. and Torczon, L. Improvements to graph coloring register allocation 1994
- [7] G. J. Chaitin, Register allocation & spilling via graph coloring, Proceedings of the 1982 SIGPLAN

symposium on Compiler construction, June 23-25,
1982, Boston, Massachusetts, United States

- [8] COINS Project COINS バックエンド部の実装
<http://www.coins-project.org/COINSdoc/backend/backend-frame.html>