静的単一代入形式上で通常形式部分冗長除去を実現する汎用的手法

今橋 孝典 † 伊藤 陽 †* 佐々 政孝 †
Takanori IMAHASHI Yo ITO Masataka SASSA

† 東京工業大学大学院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
imahash3@is.titech.ac.jp ito1@is.titech.ac.jp sassa@is.titech.ac.jp

静的単一代入（SSA）形式は様々な最適化が見通しよく行えるとされる形式であるが、最適化の一つである部分冗長除去（PRE）を SSA 形式上で実現するのは一般には容易ではない。既存の手法では PRE を SSA 形式上で実現するために特別なデータ構造などを用いて複雑な処理を行っている。本研究では、ある性質を持った SSA 形式である CSSA 形式の特徴と phi congruence class というものを利用した、通常形式の PRE アルゴリズムを SSA 形式に適用する一般的な手法を提案する。これにより、代表的な PRE アルゴリズムである Lazy Code Motion を SSA 形式に移植した。

1 はじめに

1.1 背景

部分冗長除去 (Partial Redundancy Elimination, 以後 PRE と略称することもある) は、共通部分式除去とループ不変式移動の効果を含んだ効果的な最適化である。PRE は、Morel らによって最初のアルゴリズムが提案され [12]、その後も多くの改良されたアルゴリズムが提案されている [5, 6, 7, 10, 11, 14]。

これらの研究の中で、PRE を**静的単一代入形式 (Static Single Assignment Form, 以後 SSA 形式と呼ぶ)** 上で行おうという試みがある。SSA 形式は最適化に適したプログラムの表現形式であり、近年盛んに研究が行われている。しかし、PRE を SSA 形式上で行おうとすると、

- 通常形式上では同一の変数であったものが名前替えにより異なる変数名になることがある。
- SSA 形式での変数には満たすべき条件があるので、異なるブロックにコードを移動する際に変数をそのまま移動できない。

といった困難がある（詳細は次節で述べる）。

本研究では、これらの問題を解決し、PRE アルゴリズムを SSA 形式上で実現する汎用的手法（以後、本手法という）を提案する。

1.2 本手法の概要

本手法の概要を説明する。

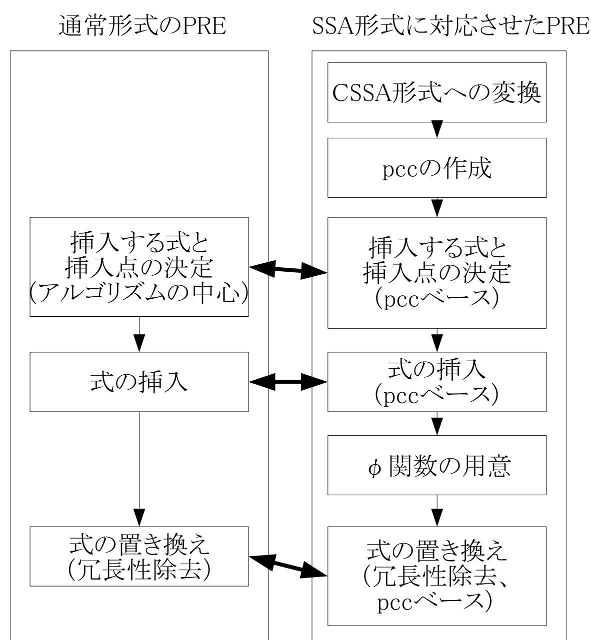

図 1: 通常形式上の PRE と SSA 形式上の PRE

一般的な PRE アルゴリズムの手順を図示すると図 1（左）のようになる。このうち、アルゴリズムの中心となるのは「挿入する式と挿入点の決定」の部分である。

一方、通常形式での PRE アルゴリズムを本手法

*現在は NEC

によって SSA 形式に対応させると、その手順は図 1（右）のようになる。このうち、元の PRE アルゴリズムに対応している部分は「挿入する式と挿入点の決定」と「式の挿入」、「式の置き換え」の三つであるが、後者二つは「$t = a + b$」の挿入や、「$\ldots = a + b$」の「$\ldots = t$」への置き換えであって、どの PRE アルゴリズムでも同じである。よって、元の PRE アルゴリズムに依存するのは実質「挿入する式と挿入点の決定」のみとなる。この処理は、変数の字面の情報の代わりに pcc（後述）の情報を用いることで、アルゴリズムの本質を変えることなく SSA 形式に対応させることが出来る。よって、図 1（左）の手順に沿うような PRE アルゴリズムであれば本手法を用いて SSA 形式に対応させることが出来る。

以下では PRE と SSA 形式について説明した後、SSA 形式上での PRE のそれぞれのステップについて詳細を述べる。その際に、図 2 にあるプログラムを例として用いる。このプログラムに本手法を適用し、PRE が実現される過程を見ていく。

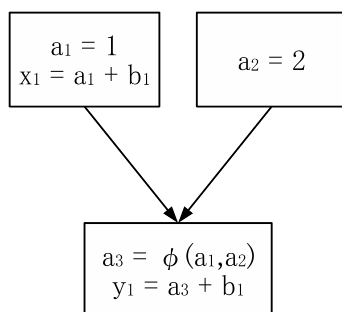

図 2: PRE 適用前のプログラム

2 部分冗長除去と SSA 形式

本節では、本研究の前提となっている部分冗長除去と SSA 形式、及び SSA 形式上での PRE の問題点について述べる。

2.1 部分冗長除去

プログラムの実行中には、ある式の値が既に計算されているにも関わらず、再びその式の値を計算し直すパスが存在することがある。PRE は、そのような冗長な計算を既に計算した計算結果で置き換える最適化の手法である。図 3 はその最も簡単な場合であり、図（左）に対して冗長性除去を行うと図（右）のようになる。

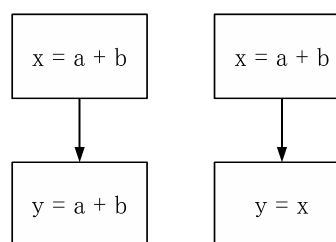

図 3: 簡単な冗長性除去の例

一般の部分冗長除去はその名が示す通り、あるプログラムパスを通ってきた場合にのみ冗長となるような計算（部分冗長という）に対しても、冗長性を除去することが出来る。たとえば図 4（左）のようなプログラムでは、左上からのパスを通ってきたときは冗長であるが右上からのパスを通ってきたときは冗長ではない。よってこれは部分冗長である。

このようなプログラムに対しては、右上のブロックに対象となる計算を挿入する事で、どのパスを通っても冗長になる（全冗長）ようにできる。その結果、冗長性が除去できるようになる（図 4 右）。以上のように、PRE の基本的な手順は、

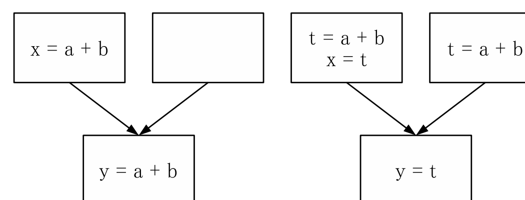

図 4: 部分冗長除去 (PRE) の例

1. プログラムに式を挿入し、部分冗長な式を全冗長にする。
2. 全冗長となった式を除去する。

である。この、挿入式と挿入点を求めることが PRE アルゴリズムの要であり、個々の部分の違いがさまざまなアルゴリズムの差であり、最適化効果の差となる。

2.2 SSA 形式

SSA 形式は、変数の定義がプログラムの字面上で唯一になるようにしたプログラムの表現形式である [1, 2, 4, 13]。静的とは、プログラムの字面上で、という意味である。変数の定義が唯一になるように変

数の名前替えを行うが、これはふつう変数に添字をつけて表す。この結果、 SSA 形式では、プログラムあるいは中間表現の上で、各変数の定義が 1 箇所だけになる（図 5)。

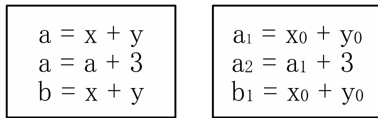

図 5: 通常形式（左）と SSA 形式のプログラム（右）

また、異なる定義の合流点にはφ関数という仮想的な関数を挿入することで定義を一つにまとめる。

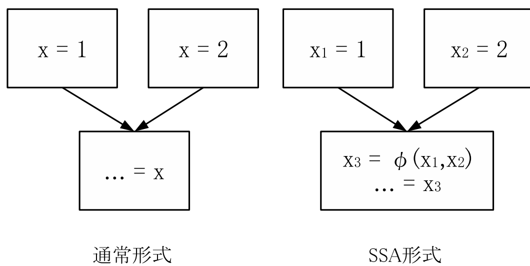

図 6: φ関数の例

図 6（右）での「$x3 = φ(x1, x2)$」は、制御の流れが左上の基本ブロックから来たときには $x3$ に $x1$ を代入し、右上の基本ブロックから来たときには $x3$ に $x2$ の値を代入する、という意味である。

SSA 形式は、プログラミング言語処理における最適化に適しているとされるプログラムの表現形式であり、近年盛んに研究が行われている。

2.3 SSA 形式上での PRE

PRE は強力な最適化であるが、これを SSA 形式上で実現するのは以下に示す二つの理由のため容易ではない。

- 通常形式上では同一の変数として扱えたものが、SSA 形式上では添え字付けのため異なる変数と認識され、「同一な式」の検出が困難になる
- 単純に異なる基本ブロックにコード移動を行うと、SSA 形式が満たすべき条件が守られなくなる可能性がある

たとえば、図 7（左）の「$a + b$」は PRE の対象となるものであるが、図 7（右）のように SSA 形式に変換すると、2 つの「$a + b$」が字面上では別々の形になってしまうため同一の式で冗長である事が判別できなくなってしまう。これが一つめの問題である。

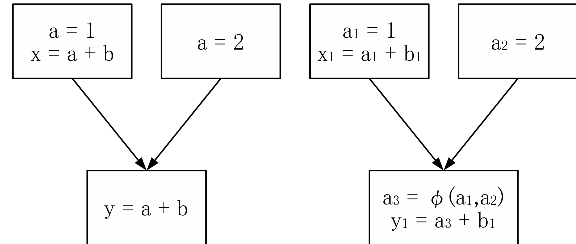

図 7: SSA 形式と PRE

また、仮に「$a1 + b1$」と「$a3 + b1$」が同じだと認識できても図 7（右）の下のブロックの「$a3 + b1$」を右上のブロックに挿入することは出来ない。何故なら、このままのコードを挿入すると、$a3$ の使用が $a3$ の定義よりも先にきてしまうからである。これが二つめの問題である。

[9, 18] などでは SSA 形式上の PRE が提案されているが、これらのアルゴリズムでは制御フローグラフとは別に特別なグラフの作成とそのグラフにおける複雑な解析が必要になっている。また、[3] のアルゴリズムでは処理中に一旦 SSA 形式から通常形式に戻ってしまう。立川は Lazy Code Motion[11] のアルゴリズムを元に、それらの問題を解決した SSA 形式上の PRE アルゴリズムを提案している [17]。しかしこの方法は後述する CSSA 形式にしか適用できない。

3 TSSA 形式と CSSA 形式

CSSA 形式とは、Cytron らのアルゴリズム [4] で SSA 形式に変換した直後の SSA 形式のことをいう。CSSA 形式の特徴として、φ関数内の変数（φ関数の左辺も含む、以後も同じ）の間に干渉が存在しない事が挙げられる。この性質が維持されているとき、φ関数内の変数を全て同じ代表変数に置き換えφ関数を除去するとプログラムの意味の等しい通常形式になる事が保証されている [16]。また、異なるφ関数内の変数同士に重なりがあるときは、両方のφ関数内の変数を全て同一の代表変数で置き換えればよいことが知られている。

このような、φ関数で結ばれた変数の集合を表す為に、phi congruence class という概念が存在する。これは直感的には、同一の代表変数に置き換えても

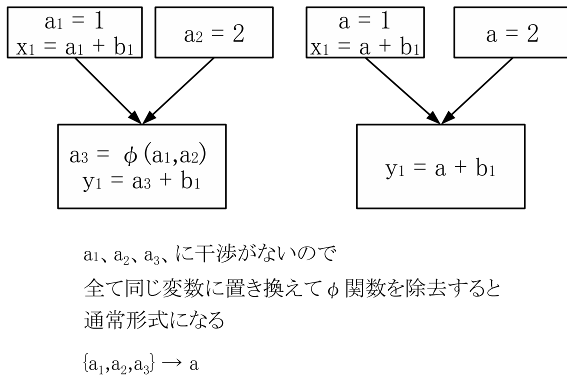

図 8: CSSA 形式の特徴

構わないような変数の集合を意味する。

しかし、SSA 形式での最適化変換を行うと、前述のような CSSA 形式の性質は一般には保たれない。つまり、φ関数の変数間に干渉が発生する。このような SSA 形式を TSSA 形式という。本手法では PRE を行う前に、[16] の手法を用いて TSSA 形式から CSSA 形式への変換を行う。以下に、[16] の手法を簡単に示す。

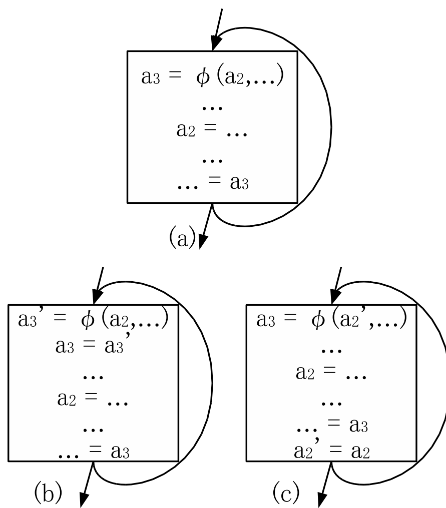

図 9: TSSA 形式から CSSA 形式への変換

図 9(a) は、変数 $a3$ と $a2$ の生存区間に重なりがある（$a3$ と $a2$ が干渉する）。このような場合、コピー文を挿入して図 9(b) あるいは図 9(c) のように変形することでφ関数内の変数間の干渉を無くすことができる。図 9(b) は、φ関数の左辺の生存区間が最小になるように変形した例であり、図 9(c) は、φ関数の引数の生存区間が最小になるように変形した例である。

4 Phi congruence class

phi congruence class(以後 pcc と略称することもある) とは、φ関数で結ばれた変数の集合である。たとえば図 8（左）では「$a3 = \phi(a1, a2)$」によって $a1$、$a2$、$a3$ がφ関数で結ばれ、同一のクラスに属すことになる。その結果、図 8 の phi congruence class は $\{a1, a2, a3\}$ となる。

一つの変数が複数のφ関数に属す場合、phi congruence class は、プログラム中の各φ関数で結ばれた変数を同じクラスにし、最後に共通部分を持つクラスをマージしたものである。その簡単な例を図 10 に挙げる。φ関数内の変数はそれぞれ $\{x, y, z\}$ と $\{u, x, w\}$ だが、x が双方に共通しているため、二つの phi congruence class がマージされ、最終的な phi congruence class は $\{x, y, z, u, w\}$ となる。

x = φ1(y,z)
u = φ2(x,w)

φ1のpcc → {x,y,z}
φ2のpcc → {u,x,w}
マージされたpcc → {x,y,z,u,w}

図 10: phi conguence class (pcc) のマージ

前節で述べたように、CSSA 形式上では同じ phi cougruence class に属す変数を同じ代表変数に置き換えるとプログラムの意味が等しい通常形式となる。よって、「CSSA 形式上で変数同士が同じ phi congruence class に属すことは、通常形式上で同じ字面の変数であること」に対応する。この事実を利用すれば、PRE を SSA 形式上で行う時の問題の一つであった「同一な式の検出」が可能になる。つまり、同一の式かどうかを判断する際に通常形式上においては同じ変数かどうかの判断を行っていたが、CSSA 形式上では代わりに同じ phi congruence class に属す変数かどうかを判断すればよい。

5 挿入する式と挿入点の決定

挿入する式と挿入点の決定は、実装する PRE アルゴリズムが行う。ただしその際、前節で述べたように、同じ変数かどうかを判断する代わりに同じ phi congruence class に属するかどうかを判断するよう、PRE アルゴリズムを変更する。

6 式の挿入

部分冗長な式を発見すると、PRE アルゴリズムは前節で求められたプログラムの挿入点に式を挿入し、部分冗長な式を全冗長にする。また、元の計算の前にもやはり一時変数に式を代入するための文が挿入される。

このような式の挿入が行われる際に、SSA 形式の満たすべき条件が問題になる。つまり、単純に式を挿入すると、変数の使用が定義に先行してしまう恐れがある。そのため、挿入する際には、挿入しても問題のない変数名に変数を書き直す必要がある。本手法では式の挿入を以下の例のように行う。

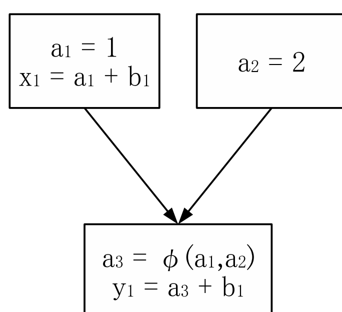

図 11: PRE 適用前のプログラム（再掲）

ここでは、図 11 のプログラムで「$a3+b1$」を左上のブロックの「$x1 = a1+b1$」の直前に挿入する場合を考える。まず、式のそれぞれの変数に対して、挿入点からその点を支配する点を上向きに辿っていく。そしてその変数と同じ phi congruence class に属す変数の定義を見つけたら、その見つかった変数を挿入する式の変数とする。この例だとまず変数 $a3$ について、同じ phi congruence class に属する変数の定義を探す事になる。その結果、$a3$ と同じ phi congruence class に属する $a1$ の定義「$a1 = 1$」が見つかる（図 12）。よって、式を実際に挿入する際には $a3$ を $a1$ と書き直す事になる。

また、$b1$ については図の中には記されていないが、上方に $b1$ の定義文があるとする。すると、$b1$ が挿入すべき変数となる。以上から、左上のブロックには、書き直された式「$a1+b1$」を挿入すればよいことがわかる。

ここまでの例では左上のブロックにのみ挿入を行ったが、通常の PRE アルゴリズムだと、右上のブロックにも式を挿入しようとする。そこで右上のブロックの「$a2 = 2$」の直後に挿入点があるとして同様の

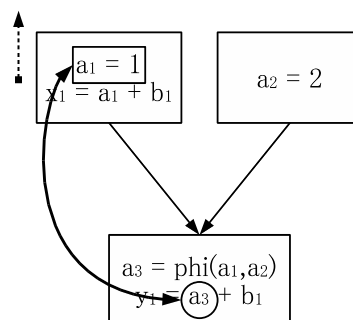

図 12: 同じ pcc に属す変数の定義の探索

処理を適用すると、最終的にプログラムは図 13 のようになる。尚、一時変数は、まだ使用されていない $t1$、$t2$ を使った。

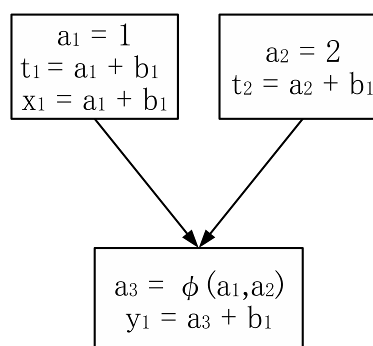

図 13: 式を挿入した直後のプログラム

7 φ関数の用意

一時変数の挿入の後は、その挿入した一時変数のためのφ関数を用意する。

7.1 φ関数の作成

まず、最小 SSA を作成するアルゴリズム [4] に従って、前節で $ti = aj + bk$ の形の式を挿入したブロックの支配辺境に、一時変数のためのφ関数を作成する。φ関数の左辺には、まだ使われていない一時変数の名前（後述の例では $t3$）を付ければよい。また、右辺の変数にはこの時点では仮の名前を付けておく。なぜなら、φ関数の右辺には、この手続きで挿入される他のφ関数の左辺が挿入されるかもしれないの

で、一通りφ関数を挿入した後でないと右辺を正しく決める事ができないからである [8]。図 13 のプログラムに対して、一時変数のためのφ関数を作成すると図 14 のようになる。

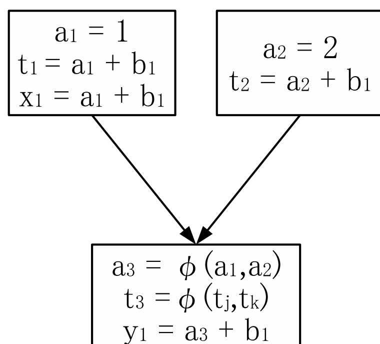

図 14: 一時変数のφ関数を作成したプログラム

7.2 φ関数内の変数の名前替え

φ関数を一通り挿入した後で、φ関数の右辺の変数を適切な名前に書き換える処理を行う。

例えば図 14 の式「$t3 = \phi\ (tj, tk)$」の tj を置き換える場合を考える。この場合、問題のφ関数の左上のブロックから、支配するブロックを辿っていき、前節で挿入した式か、本節前半で作成したφ関数を探す。そしてどちらかが見つかるとその左辺の一時変数でφ関数内の変数を置き換える。この例だと、左上のブロックの式「$t1 = a1 + b1$」が前節で挿入した式であるので、左辺の $t1$ で tj を置き換える。tk を置き換える場合は、右上のブロックから同様に探索を行う。この例の場合は式「$t2 = a2 + b1$」が見つかるのでその左辺 $t2$ で tk を置き換える。これらの探索の様子を図示すると図 15 のようになり、名前替えの結果、最終的に得られるプログラムは図 16 のようになる。

8 冗長となった式の除去

冗長な式の除去は、実際には式の一時変数への置き換えである。その際、置き換える一時変数を決める必要がある。対象とする式一つに対し、挿入された一時変数は全て同じ phi congruence class に属している。よって、置き換える式の存在する点から、その点を支配する点を上に辿っていき、挿入された一時変数が属する phi congruence class に属す変数の定義を見つけたら、その変数で式を置き換える。

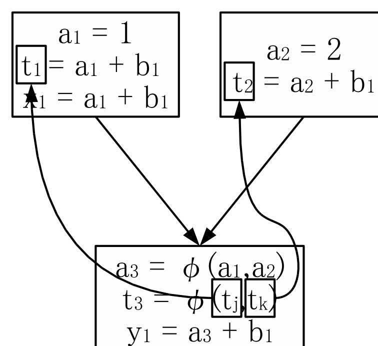

図 15: φ関数内の変数の名前替え

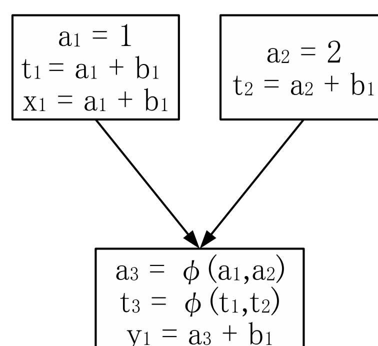

図 16: φ関数の用意が完了したプログラム

たとえば図 16 では挿入された一時変数の phi congruence class は $\{t1, t2, t3\}$ である。下のブロックの「$a3 + b1$」を一時変数に置き換えたいときは、ここから支配関係を上に辿ると一時変数と同じ phi congruece class に属す変数 $t3$ の定義「$t3 = \phi\ (t1, t2)$」が見つかるので、$t3$ で置き換える。左上のブロックの「$a1 + b1$」を置き換えたいときは、同様にして $t1$ の定義「$t1 = a1 + b1$」が見つかるので $t1$ で置き換える。

以上の処理によって SSA 形式上での PRE は達成され、最終的に得られるプログラムは図 17 のようになる。

9 実験

本研究では実験として、代表的な PRE アルゴリズムである Lazy code motion [11] を SSA 形式に対応させた。実験には COINS の SSA 形式最適化コンパイラ用モジュール [15] を用い、Sun Blade 1000 (UltraSPARC III) で実行時間を測定した。評価した最適化はすべて SSA 形式上のものである。

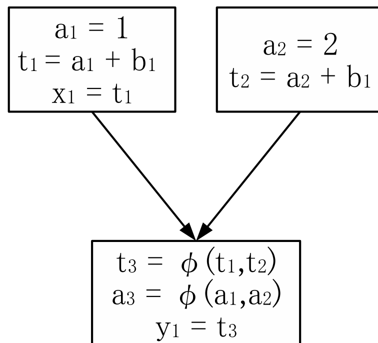

図 17: SSA 形式上の PRE の結果

最適化としての効果を確認するためのベンチマークプログラムとして、SPEC CINT2000 の 6 個のベンチマークを使用して、各最適化を行なった。

9.1 適用した最適化列

まず本手法で実装した SSA 形式上の Lazy code motion の単体での最適化効果を調べるため、以下の最適化列を適用し、実行時間を計測した。

- no-opt :最適化無し
- LCM :SSA 形式 Lazy code motion + 不要命令除去

最適化を何も適用しない「no-opt」と、SSA 形式 Lazy code motion のみを行う「LCM」とで実験を行った。ただし、一般的に最適化は後のフェーズで不要命令除去を行うことを前提としているため、「LCM」では最後のフェーズで不要命令除去を行っている。

また、Lazy code motion を他の様々な最適化と組み合わせた時の効果を調べるため、以下の最適化列の効果を調べた。

- O2 :共通部分式除去→ 定数伝播→ ループ不変式移動→ 演算の強さの軽減→ ループ不変式移動→ 定数伝播→ コピー伝播→ preqp → 定数伝播→ 不要命令除去
- O2-LCM :共通部分式除去→ 定数伝播→ ループ不変式移動→ 演算の強さの軽減→ ループ不変式移動→ 定数伝播→ コピー伝播→ SSA 形式 Lazy code motion → 定数伝播→ 不要命令除去

「O2」とは現在 COINS でよい結果が得られるとされている最適化列であり、O2 オプションを指定すると上記の最適化列が実行される。その中にある preqp とは、滝本の質問伝播に基づく大域値番号付けと部分冗長除去である [15]。「O2-LCM」は、「O2」の中の preqp を SSA 形式 Lazy code motion に置き換えたものである。

9.2 結果と評価

実験結果は図 18 のようになった。

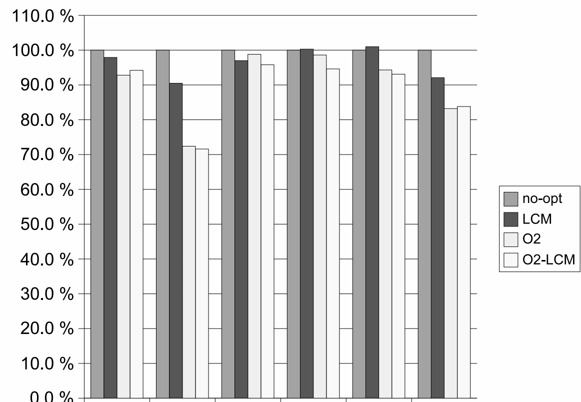

図 18: 実験結果 (no-opt との比率)

SSA 形式 Lazy code motion を単体で適用した LCM では、最大で約 10 %の効果があった。また preqp は部分冗長除去を行う最適化であるが、O2 と O2-LCM とを比較すれば、preqp と SSA 形式 Lazy code motion がほぼ同等の効果を挙げていることがわかる。以上から、本手法で実装した SSA 形式 Lazy code motion が正しく部分冗長除去の効果を発揮できていることが確認できた。

10 まとめ

静的単一代入形式 (SSA 形式) 上で部分冗長除去 (PRE) を行うことは困難であったが、本研究では PRE を SSA 形式上で実現する手法を提案した。それは、SSA 形式上での変数名の同一性の判断や式の移動などに、CSSA 形式と phi congruence class(pcc) の特徴を利用するというものである。

この手法は汎用的なものであり、今回実装した Lazy code motion 以外の多くの通常形式上の PRE アルゴリズムにも適用可能である。特に、図 1（左）の手順に沿ったアルゴリズムならば、「挿入する式と挿入点の決定」の処理さえ正しく pcc ベースのものに変換出来れば、SSA 形式に対応させる事が出来る。例

えば [8] ではこの手法を用いて実行時情報を利用した PRE を SSA 形式上で実現させている。

PRE を SSA 形式上で実現出来れば、逆変換のオーバーヘッド無しに SSA 形式上での最適化列の中に PRE を組み込む事が可能になる。また、ビットベクタを用いた解析時間の短縮が見込まれるなどの有用性も挙げられる。

参考文献

[1] B. Alpern, M. N. Wegman, and F. K. Zadeck. Detecting equality of variables in programs. In *POPL '88: Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 1–11, New York, NY, USA, 1988. ACM Press.

[2] A. W. Appel and J Palsberg. *Modern Compiler Implementation in Java 2nd ed.* Cambridge University Press, 2002.

[3] Preston Briggs and Keith D. Cooper. Effective partial redundancy elimination. In *SIGPLAN Conference on Programming Language Design and Implementation*, pp. 159–170, 1994.

[4] Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems*, Vol. 13, No. 4, pp. 451–490, October 1991.

[5] D. M. Dhamdhere. A fast algorithm for code movement optimisation. *SIGPLAN Not.*, Vol. 23, No. 10, pp. 172–180, 1988.

[6] D. M. Dhamdhere. Practical adaption of the global optimization algorithm of morel and renvoise. *ACM Trans. Program. Lang. Syst.*, Vol. 13, No. 2, pp. 291–294, 1991.

[7] Dhananjay M. Dhamdhere. E-path_pre: partial redundancy elimination made easy. *SIGPLAN Not.*, Vol. 37, No. 8, pp. 53–65, 2002.

[8] 伊藤陽, 佐々政孝. 実行時情報を利用した部分冗長除去と SSA 形式への適用. 日本ソフトウェア科学会第８回プログラミングおよびプログラミング言語 ワークショップ (PPL2006) 論文集, pp. 170–181, 2006.

[9] Robert Kennedy, Sun Chan, Shin-Ming Liu, Raymond Lo, Peng Tu, and Fred Chow. Partial redundancy elimination in SSA form. *ACM Transactions on Programming Languages and Systems*, Vol. 21, No. 3, pp. 627–676, 1999.

[10] J. Knoop, O. Rüthing, and B. Steffen. Lazy code motion. In *Proceedings of the ACM SIGPLAN '92 Conference on Programming Language Design and Implementation*, Vol. 27, pp. 224–234, San Francisco, CA, June 1992.

[11] Jens Knoop, Oliver Rüthing, and Bernhard Steffen. Optimal code motion: Theory and practice. *ACM Transactions on Programming Languages and Systems*, Vol. 16, No. 4, pp. 1117–1155, July 1994.

[12] E. Morel and C. Renvoise. Global optimization by suppression of partial redundancies. *Commun. ACM*, Vol. 22, No. 2, pp. 96–103, 1979.

[13] 中田育男. コンパイラの構成と最適化. 朝倉書店, 1999.

[14] Vineeth Kumar Paleri, Y. N. Srikant, and Priti Shankar. A simple algorithm for partial redundancy elimination. *SIGPLAN Not.*, Vol. 33, No. 12, pp. 35–43, 1998.

[15] 佐々研究室. 静的単一代入形式最適化システム外部仕様書, 2006. http://www.is.titech.ac.jp/˜sassa/coins-www-ssa/japanese/ssa-external-japanese.pdf.

[16] Vugranam C. Sreedhar, Roy Dz-Ching Ju, David M. Gillies, and Vatsa Santhanam. Translating out of static single assignment form. In *SAS '99: Proceedings of the 6th International Symposium on Static Analysis*, pp. 194–210, London, UK, 1999. Springer-Verlag.

[17] 立川英. 静的単一代入形式上の部分冗長除去. 東京工業大学 大学院情報理工学研究科 数理・計算科学専攻 修士論文, 2004.

[18] 滝本宗宏, 原田賢一. 拡張値グラフに基づく効果的な部分冗長除去法. 情報処理学会論文誌, Vol. 38, No. 11, pp. 2237–2250, 1997.