部分的無効コードの効率的除去法

3J－5

滝本宗宏　原田賢一
慶應義塾大学理工学部

1　はじめに

コード最適化の分野において，制御の流れによっては，実行されないコードを除去するための手法が最近注目されている．このような制御依存で無効になるコードは部分的無効コード (Partial Dead Code)[BC][KOS] と呼ばれる．部分的無効コード除去において，特に，二次的効果を考慮した徹底的な解析を行う手法として，J. Knoop, O. Rüthing, B. Steffen の研究が発表されている．この手法は，コード降下 (Code Sinking) と無効コード除去を交互に行うプログラム変形を，プログラムが不変になるまで，適用することによって実現されている．

2つの相互に依存するアルゴリズムの形により，非常に大きな計算コストを必要とする．

本稿では，この二次的効果を含む部分的無効コード除去を効率的に行う手法を提案する．コストを大きくする，コード降下と無効コード除去の相互依存性を排除し，それぞれ，独立に適用する．2つの解析の間で保存されるべき依存情報は，グラフの形 (拡張値グラフ) で保持する．コード降下を行い，拡張値グラフにより，プログラムの意味を変えないよう，降下の範囲を修正したのち，無効コード除去を適用する．

2　二次的効果

新たな部分的無効コードを生成するために重要なコード降下は，プログラムの意味保存のために，次のいずれかの場合にブロックされる．コード降下の対象を $\alpha \equiv x := t$ として，

1. t のオペランドが後続コードによって，変更される場合
2. 変数 x が後続コードによって，変更される場合
3. x が後続コードに使用されている場合

これらの条件による降下の抑制は，ブロックしている後続コードの降下，また，除去によって，解除される．

このような，コード降下と無効コード除去の間に存在する二次的効果は，次の4つにまとめることができる．

降下 - 除去効果 コード降下が新たな部分的無効コードを生む．

降下 - 降下効果 ブロックしている後続コードの降下によって，さらなる降下が可能になる．

除去 - 降下効果 ブロックしている後続コードの除去によって，さらなる降下が可能となる．

除去 - 除去効果 後続コードの除去によって，現在のコードが除去の対象となる．

これらの二次的効果は，降下のブロックの存在によって，まさに，二次的にのみ得られるものである．二次的効果を含む範囲で，一度にコードを降下させ，その後に，無効なコードを除去するには，コード降下と，コード相互におけるプログラムの意味の保存を，独立に行う必要がある．

3　SSA 形式によるブロックの回避

本研究において，プログラムはSSA 形式 (Static Single Assignment Form) で表されているものとする．SSA 形式は，ある変数に対する定義が唯一存在するように変形したプログラム形式のことである．異なった定義が制御の流れに依存し到達するような，定義の結合部分には，フローグラフの各先行節から到達する変数を引数として持ち，新たな変数への代入を行う ϕ - 関数を導入する．

SSA 形式に変更する意義は，各定義に対して，異なった代入先変数が用意されるため，コード降下のブロック条件のうち，1. と 2. の後続コードの影響を考慮する必要がなくなる．また，このSSA 形式にプログラムを変形しておくことによって，次に用意する拡張値グラフの前提となる値グラフを容易に構築することができる．

4　拡張値グラフとその変形

SSA 形式により，コード降下のブロック条件の 1. と 2. を回避することができることを示した．しかし，最後の条件 3. は，ある演算が行われる前に，そのオペランドの計算が行われなければならないという，計算実行の半順序保存の規定である．

この条件を満足するように，プログラムを修正するためには，演算子とオペランドの依存関係を保存しておく必要がある．このような依存関係を保持する構造として，値グラフが存在する．

値グラフは，変数の定義と使用を辺で結んだ形の有効グラフである．辺は，使用から定義に向かう方向を持っている．$X \leftarrow Y$ のような，コードは，変数 X の定義として，Y の定義を用いる．値グラフはSSA 形式から，容易に構築することができる．

コード降下において，同じパターンを持つ代入文が，全ての先行節から，降下してきた場合，さらに，降下を続けることができる．しかし，複数の先行節から，異

Efficient partial dead code elimination
Munehiro Takimoto, Ken'ichi Harada
Keio University
3-14-1 Hiyoshi, Kouhoku-ku, Yokohama 223, JAPAN

なった右辺の定義を持つコードが降下してくる場合は，制御の結合以降のコードを表現する値グラフ節は存在しない．そこで，次のような，値グラフの変形を行う．

値グラフ VG 上に，ある ϕ- 節 (ϕ - 関数に相当する節) V_ϕ が存在して，n 個の各節 $V_i(for\ i = 1,2,\cdots,n)$ に依存しているとする．すべての i について，V_i のラベル (ϕ- 関数ではない) は同じであり，各 V_i は m 個の依存先 $V_{ij}(for\ j = 1,2,\cdots,m)$ を持つとする．

ある k が存在して，各 i において，節 V_{ik} を i 番目の依存先とする，V_ϕ と同じ ϕ - 関数を持つ節が存在するか，全ての i について，V_{ik} が同じ節である場合，V_i と同じラベルをもつ節 V' を生成して，V_ϕ と置き換える．各 V' の j 番目の依存先を，各 V_i が j 番目の依存先としている V_{ij} を各依存先とする ϕ - 節，または，全ての i に対して，V_{ik}(ある k において) が同じであった場合はその節とする．

この変形は，右辺が異なった定義を持ち，パターンが同じ代入文が，全ての先行節から降下してくる状況を，値グラフに置き換えたにすぎない．

値グラフとその変形を示したが，実際には，値グラフを拡張した，拡張値グラフというものを用いる．値グラフを，ただ変形してしまうと，元の値グラフには戻れなくなってしまう．降下がどの程度起こるか，わからない状況で，初期の依存グラフが崩すわけにはいかない．コード降下の全ての状況における依存グラフとして，働くように，以前の構造を保存する戦略を採る．

拡張値グラフは，値グラフを節によって，まとめた形をしている．各節は値グラフを 2 つまで保持できる．拡張値グラフの辺は，まとめた値グラフの辺に一致し，依存先は，拡張値グラフ節である．

各 ϕ 節を含む拡張値グラフ節は上の変形により，実行可能関数 (演算子) となるが，その節は，ϕ 節と共に保持する．

5 データフローグラフの構築

各 ϕ に関して，変形を終了した拡張値グラフを用いて，コード降下のデータフロー解析のベースとして使用するデータフローグラフを構築する．構築は次の手順で行う．

1. 拡張値グラフ節のうち，初期に ϕ 節を持っていた節で，変形後，実行可能関数を持つようになったものに関して，その ϕ - 関数が置かれているフローグラフ節とその先行節の間に，ϕ - 関数が結ぶ定義用のデータフローグラフ節を生成し，辺で結ぶ．
2. 各拡張値グラフ節に相当する定義をプログラムの出口から，後向きに，節を作り，辺でつないでいく (ϕ - 関数によって挿入した節も含めて)．

1. はパターンが同じ代入式を互いに結び合わせる役割をする．1. により，拡張値グラフの各節に対して，データフローグラフ節を生成しながら，辺で結んでいくだけで，プログラム全体に渡るグラフを構築することができる．

6 コード降下と修正

この後，データフローグラフ上で，コード降下のデータフロー解析を行う．コード降下を $DELAYED$ として，データフロー式の中心は次のものである．

$$DELAYED_n =_{df} \begin{cases} false & if\ n = s \\ \prod_{m \in pred(n)} DELAYED_m & otherwise \end{cases}$$

ここで，s, e, n, m はデータフローグラフ節であり，特に，s は入口節であり，e は出口節である．

ただ，このデータフロー解析をしただけでは，オペランドの定義がその使用を飛び越えるということが起きる場合がある．つまり，コード降下をブロックする条件 3. である．ここで，拡張値グラフを用いる．各拡張値グラフ節は，対応するデータフローグラフ節を持っている．同じ基本ブロック上のデータフローグラフ節に関して，拡張値グラフの依存関係により，$DELAYED$ に修正を加える．修正に使用する式は次のようになる．

$$DELAYED_v(N) =_{df} \prod_{v'\ depends\ on\ v} \begin{matrix} \text{初期のプログラ} \\ \text{ムにおける } v' \text{ に} \\ \text{あたる定義から，} \\ \text{入口までの範囲} \end{matrix} \vee DELAYED_{v'}(N)$$

ここで，N は同じ基本ブロックのデータフローグラフ節であることを意味する．v, v' は拡張値グラフ節である．

この修正によって，使用を飛び越えるかたちで，$DELAYED$ $=true$ となっていた，定義を $false$ にしていくことができる．

7 無効コード除去

コード降下が終了した後，$DELAYED$=$ture$ の最も遠い (入口から) 場所に挿入を行う．元から，プログラムに存在したコードに関しては，それを消去する．今，プログラムは SSA 形式になっているので，ϕ - 関数を，先行節に，各引数を右辺とする代入式として挿入することによって，除去する．

最後に無効コードの除去を行う．本研究における無効コードの除去は，実際は，弱体コード (Faint Code) 除去の手法を用いる．弱体コードとは，その代入先の変数が，無効コードまたは，弱体コードのみで使用されているものである．これは [KOS] において使用されている弱体コード除去のアルゴリズムを使用する．これは，後向きのデータフロー解析である．

参考文献

[BC] P. Briggs and K. D. Cooper. Effective Partial Redundancy Elimination. SIGPLAN '94 ACM Conf. PLDI, 1994, 159-170.

[KOS] J. Knoop, O. Rüthing and B. Steffen. Partial Dead Code Elimination. SIGPLAN '94 ACM Conf. PLDI, 1994, 147-158.