部分冗長除去に基づく実践的な大域命令スケジューリング

及川 亮太郎† 滝本 宗宏††

コンパイラの 1 つである最適化である命令スケジューリングを部分冗長除去法に基づいて行う手法を提案する. 本手法は, 実行パス上の命令数を増やさず, 効果のない命令の移動を行わないことを保証しながら, プログラム全体に対して, 命令スケジューリングを適用することができる. またループスケジューリングの1つであるループシフティングの効果を持つのでループ構造を特別扱いせずに大域命令スケジューリングが可能である.

Practical Global Instruction Scheduling Based on Partial Redundancy Elimination

RYOTARO OIKAWA† MUNEHIRO TAKMOTO††

1. はじめに

命令レベル並列性を高める**命令スケジューリング**は, 命令を並列に実行することができる VLIW プロセッサやスーパースカラプロセッサの性能向上において有効なコード最適化手法である. 命令スケジューリングは, 分岐を含まない基本ブロックをターゲットとした局所的命令スケジューリングが中心だが, 解析範囲が狭いため並列性の向上には限界がある. 一方, 解析対象がプログラム全体である**大域的命令スケジューリング**[4]は, 基本ブロックを超えたスケジューリングを行うので, プログラムの意味を保存するための**補償コード**[4]の挿入や, ループの認識などが必要となり複雑だが, 局所的命令スケジューリングよりも広範囲にわたって並列性を抽出できる利点がある.

2. 従来手法

2.1. 投機的スケジューリング

ある操作を, 異なる実行パスにスケジュールすることで並列性を更に高めることができる場合がある. **投機的スケジューリング**(speculation)[1]は, そのような投機的実行を許す大域的命令スケジューリングの一つである.

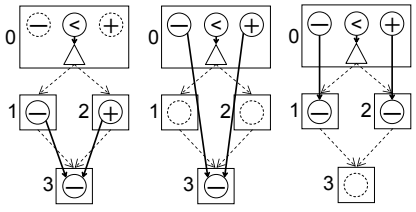


図 1 投機的スケジューリング

図 1 に示す**制御フローグラフ**は投機的スケジューリングとその効果を示している. 矩形は制御フローグラフ節を表し, 矢印は制御フローを意味する. 演算子ラベルを持つ円形は操作を表し, 点線の円形は**空きリソース**(idle resource)を表している. また, 実線矢印は, **データフロー**(data flow)を表す. 図 1(a)の節 0 において操作＋の資源が空いているので, 図 1(b)のように, 節 2 の操作＋を節 0 にスケジュールできる. このようなスケジューリングは, 実行パス 0,1,3 に新しく操作＋を導入するので, 投機的スケジューリングである. 節 1 の－も同様に, 節 0 のもう 1 つの空きリースに投機的にスケジュールできる. 更に投機的スケジューリングによって移動元には空きリソースが新たに発生するので, 図 1(b)の節 3 の操作－を図 1(c)に示すように節 1, 2 にスケジュールできることを可能にする.

2.2. 動的 CSE

投機的スケジューリングを適用すると, 冗長な操作を生成することがある. 図 3(a)に示すフローグラフにおいて, 節 2 の操作 b+c を節 0 に投機的スケジュールする. その結果, 図 3(b)において, 節 0 に対して節 5 の操作 b+c が冗長になっている. この冗長な操作は, 基本的なコード最適化である共通部分式除去(common sub-expression elimination, 以下 CSE と呼ぶ)を適用することにより, 図 3(c)の節 5 のように除去することができる.

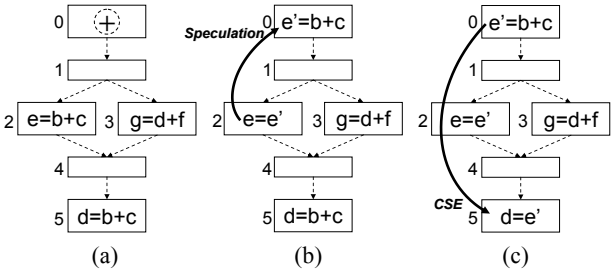


図 3 Dynamic CSE

このように, スケジューリングを行いながら, 過程で生じた共通部分式を CSE で除去していく手法は**動的 CSE**(Dynamic CSE)と呼ばれ, 実行パスを更に短くする可能性がある.

3. 部分冗長除去

ある式の出現が, その出現を含む幾つかのパスで冗長であるが, 他の実行パスでは冗長でない場合, その式は, **部分冗長**(partially redundant)であると言う. **部分冗長除去**(partial redundancy elimination, 以下 PRE と呼ぶ)[3,4]は, 冗長でないパスに同じ式を補償コードとして挿入することによって, 全冗長な式に変換し, 除去することができる. PRE において, 式を挿入し, 元の式を削除するという過程は, **コード巻上げ**(code hoisting)に相当する. 図 4(a)で, 式 exp は実行パス 0, 2 において冗長であるが, パス 1, 2 においては冗長ではない. このような部分冗長性は, 図 4(b)に示すような式の巻上げによって除去できる.

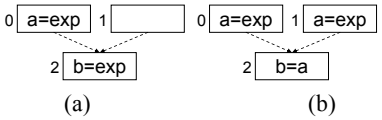


図 4 部分冗長除去

†東京理科大学大学院理工学研究科
††東京理科大学理工学部

更に， PRE のコード巻上げは，ループ不変コードを持つループに適用した場合，ループ不変コードをループの外に出す効果を持つことが知られている．図 5(a)の節 1 でループ不変な式 b+c は図 5(b)で示すように，ループの外に出される．

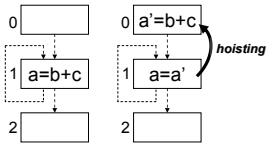


図 5 ループ不変コードの巻上げ

PREは，MorelとRenvoiseによって提案され，その後，多くの改良がなされた．現在の PRE は，データフロー方程式によって，次の条件を満たすことを補償する．

1. 実行パス上の式の数を増やさない
2. 冗長な式を除去する効果のないコード巻上げを行わない

4. PRE に基づく大域命令スケジューリング

PRE に基づく大域命令スケジューリングのアルゴリズムを示し，本手法の枠組みで，どのようにループシフティングが実現できるのかを示す．

4.1. アルゴリズム

命令スケジューリングは空きリソースに命令をスケジュールすることで実行パスの長さを短くするコード最適化である．本手法は空きリソースと移動する命令の選択方法に依存しないので，それらは適切な方法で決定されると仮定する．

1. 空きリソースと移動可能な操作を見つける
2. 空きリソースへ投機的に操作のコピーを配置
3. PRE のデータフロー解析フェーズを実行
4. コピー元が冗長かつコピー先が冗長でない時，PRE の挿入と除去によって，プログラムを変形する
5. 4 の条件が成り立たなければ，コピー先を削除

手順 2 で行った投機的なコードのコピーが，手順 4 の条件によって，スケジューリングの結果が操作の数を増やさず，実際にスケジュール可能であると判定できたとき，補償コードを自動的に挿入し，コードの巻上げを行う．一般に，補償コードは，更なる命令スケジューリングの適用によって，空きリソースに割り当てられる可能性がある．

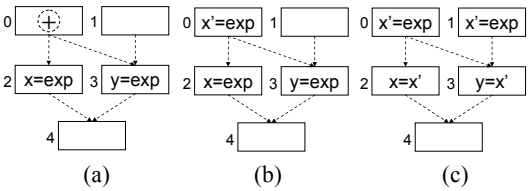


図 6 PRE に基づく大域命令スケジューリング

図 6(a)の節 0 に空きリソース，節 2 に移動可能な操作 exp があると考える．手順 2 にしたがって，exp を空きリソースに投機的にコピーすると，図 6(b)のようになる．図 6(b)において，節 2 の操作 exp は冗長であり，節 0 の exp は冗長ではない．また節 3 の操作 exp は部分冗長である．そこで手順 3 でのデータフロー解析の結果に基づいて，補償コードの挿入と，冗長な操作の除去を行うと，図 6(c)のように変形できる．

4.2. ループシフティング

ループシフティング（loop shifting）はループを対象とする**ループスケジューリング**である．ループ内に依存を持つ操作を，ループのバック辺に沿って，スケジュールすることで，ループ本体の実行パスを短くする手法である．図 7(a)において，x=x+1 は，ループ内に依存があるので，ループの外には出せない．ここで，図 7(b)が示すように**バック辺**（back edge）に沿って空きリソースにスケジューリングすると，ループ本体の実行パスを短くできる．この際，節 0 のループ前ヘッダに，補償コードを挿入する必要がある．

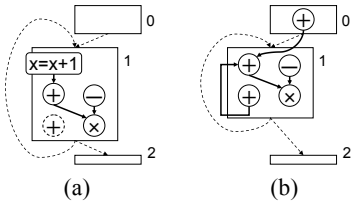
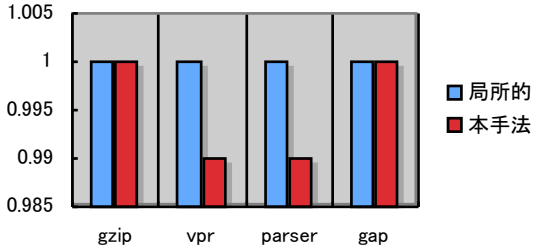


図 7 ループシフティング

ループスケジューリングは，ループ構造を抽出する必要があった． PRE に基づく大域命令スケジューリングは，このようなループシフティングを，区別することなく実現することができる．

5. 評価

本手法を，コンパイラインフラストラクチャ COINS 上で実装し，Sparc マシン上で SPEC2000 ベンチマークに対して評価を行った．ターゲットアーキテクチャは sparc で，比較対象には，COINS に備わっている局所的スケジューリングを用いた．結果として，図 XX に示すように，vpr と parser のプログラムに対して，それぞれ，1%の実行効率の向上が得られることを確認した．



6. まとめ

部分冗長除去法を用いて大域的命令スケジューリングを提案した．部分冗長除去法は実行パス上の計算を増やさず，無駄なコード移動を行わないことを保証するのでプログラムに悪影響を与えにくい．またループ構造を認識する必要なくループシフティングを実現する．

参考文献

[1] S.Gupta, R.Gupta, N.D.Dutt, and A.Nicolau,. SPARK: A Parallelizing Approach to the High-Level Synthesis of Digital Circuits, Kluwer Academic Publishers, 2004.

[2] A.V.Aho, R.Sethi, and J.D.Ullman, Compiler: Principles, Techniques, and Tools, Addison Wesley, 1986.

[3] E.Morel and C.Renvoise, Global Optimization by Suppression of Partial Redundancies, CACM, Vol.22, No.2, pp.96-103, 1979

[4] R.Allen and K.Kennedy, OPTIMIZING COMPILERS for MODERN ARCHITECTURES, Morgan Kaufmann Publishers, 2001