6K-03

コンパイラのコード最適化に基づいた レジスタとキャッシュメモリの使用の効率化

澄川 靖信 *　　滝本 宗宏 *
東京理科大学

1　はじめに

コンパイラのコード最適化は入力されたプログラムを効率の良いものへ変換するフェーズである．特に命令の実行点を変更することによって目的コードの実行効率を向上する手法として，同じ値を繰り返し計算する冗長な式を除去する**部分冗長除去**（partial redundant elimination，PRE）や，計算結果が使用されないにも関わらず不要に計算を行う式を除去する**部分無用コード除去**（partial dead code elimination，PDE）は強力な手法として知られている．しかしながら，これらの手法は，いくつかの変数の生存期間を伸長する式の挿入を行い，レジスタ割付けの際に生じるスピルを増加させることがある．スピルが生じるとロード命令とストア命令が挿入されるので，スピル増加による実行効率の低減が，式の除去による向上を上回り，結果として目的コードの実行効率を低減する原因の 1 つとなることが知られている．特に近年提案されている PRE と PDE の手法は，ある式を移動することによってその式に依存する式を移動できるようになる**副次的効果**（second order effect）を，1 回の適用で可能な限り反映する傾向がある [3, 4]．副次的効果の反映は移動する式数を増加することを意味するので，近年提案されている手法は特に多くのスピルを増加する傾向がある．

本研究では，スピルを増加させない範囲で PDE と PRE の拡張を各式に適用する手法を提案する．特にロード命令に対しては，参照先のデータがキャッシュメモリに配置されているとき，冗長性を除去するのではなく，キャッシュヒットを生じるように移動する．

Improving Utilization of Register and Cache Memory Based on Code Optimizations
*Sumikawa Yasunobu, Tokyo University of Science
*Munehiro Takimoto, Tokyo University of Science

2　入力プログラム

本手法を適用する前に，静的単一代入形式の 3 番地コードに変換されている各プログラムに対して制御フローグラフ（CFG）が作成されているとする．CFG は，文を節とする節の集合 **N**，節間の制御のフローを表す辺の集合 **E**，特別な節である開始節 **s**，終了節 **e** からなる．

3　アルゴリズム

本手法は，次の 3 つのステップからなる．1) 同じ値を計算する式に対して同じ値番号を割り付ける大域値番号付け．2) 値番号に基づく降下．3) 値番号に基づく巻上げ．本手法の降下は，PDE と同様に，計算結果が使用されない**死んでいる**（dead）式を，従来法よりも多く除去できるように，値番号の一致性を解析する．また，巻上げは，スカラー式に対しては，PRE と同様に冗長な式を除去するが，スピルを増加しない範囲で巻き上げる．ロード命令に対しては，参照先データがキャッシュメモリに配置されているとき，その冗長性は除去せずに，ヒットを生じるように移動する．以降，本手法の降下と巻上げについて説明する．

3.1　値番号に基づく降下

逆向きトポロジカルソート順序で CFG を訪問し，節 n の文 $v = exp$ の出現ごとに，v とその値番号の出現を調べるクエリを前向きに伝播する．クエリの解 *true* は，v が n で死んでいること意味し，*false* は生きている（live）を意味する．クエリは分岐点 *ifn* において，それぞれの後続節にそのコピーを伝播する．*ifn* において *true* と *false* の両方が得られたとき，*exp* を *false* を得た後続節に挿入し，*true* を返す．解 *true* は，同じ値番号の式に到達したとき，あるいは終了節に伝播されたときに得る．一方 *false* は変数の使用が出現したときに得る．最終的に，クエリを伝播した式に対して *true* が得られたとき，その式を除去する．

例：図 1(a) の節 2 の文 b_1=a_1+1 を用いて本手法の降下

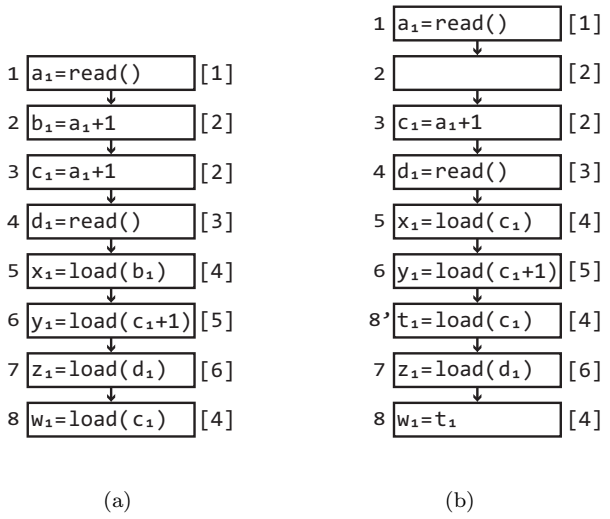

図 1　(a) 元のプログラム．(b) 本手法を適用した結果．

の効果を説明する．なお，図 1 では節の右側に，値番号を”[ ]”で囲む数字で表す．前述した文の右辺に対するクエリを前向きに伝播すると，節 3 にクエリを生成した式と同じ値番号が出現するので，b_1 を節 3 で定義する変数 c_1 に変更するコピー伝播を適用して解 *true* を得る．結果として節 2 の文を除去できる．∎

3.2　値番号に基づく巻上げ

式の演算子の実行コストを基に，演算コストの高い順にソートされたスタック *worklist* を作成する．*worklist* から取り出した式 e から順に，降下と同様に，値番号の出現を調べるクエリを後向きに伝播する．クエリの解 *true* と *false* は，それぞれ，クエリが実行経路上で冗長かどうかを意味する．*true* と *false* の両方が合流点で得られたとき，e はその点で部分冗長なので，*false* を得た節に e を挿入し，*true* を返す．解 *true* はクエリと同じ値番号が出現したときに得る．一方 *false* は，開始節に伝播されたとき，あるいはレジスタ圧力が使用可能なレジスタ数よりも多いときに得る．最終的に，e に対して *true* が得られたとき，挿入した式を基に除去する．e がロード式のとき，伝播先の節にメモリ参照式 me の存在を調べ，me の実行によって e がキャッシュヒットを生じるとき，ヒットを生じる範囲内でレジスタ圧力が最も低くなる点に式を挿入し，*true* を返す．

例：図 1(a) の節 8 の文 w_1=load(c_1) の右辺に対するクエリを後向きに伝播すると，節 6 のロード式の実行によってクエリを生成した式がキャッシュヒットを生じることを解析できる．しかしながら，節 7 のロード式によってそのデータがキャッシュから取り除かれるとき，図 1(b) に示すように，節 7 の直前に t_1=load(c_1) を挿入し，t_1 の結果を使用するように節 8 の文を変更する．∎

4　関連研究

コード移動の適用によるスピル増加を解決するために，実行回数の多い式から順に PRE を適用する手法 [2] や，各変数の生存期間が短くなるように線形計画法を用いた手法 [1] が提案されている．しかしながら，各式を大域的に移動させた場合，これらの手法によって実行効率の向上の報告はされていない．

5　まとめ

本稿では，スピル増加を防ぎながらコード移動を行う手法を提案した．本手法は，まず最初に大域値番号付けを適用し，その結果を用いる降下と巻上げを行う．特に，ロード式がキャッシュヒットを生じるとき，その冗長性を除去するのではなく，キャッシュヒットを生じるように移動する．

参考文献

[1] Gergo Barany and Andreas Krall. Optimal and heuristic global code motion for minimal spilling. In *Proceedings of the 22nd international conference on Compiler Construction*, CC'13, pp. 21–40, Berlin, Heidelberg, 2013. Springer-Verlag.

[2] Rajiv Gupta and Rastislav Bodik. Register pressure sensitive redundancy elimination. In *Proceedings of the 8th International Conference on Compiler Construction, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'99*, CC '99, pp. 107–121, London, UK, UK, 1999. Springer-Verlag.

[3] Yasunobu Sumikawa and Munehiro Takimoto. Effective demand-driven partial redundancy elimination. *Information Processing Society of Japan Transactions on Programming*, Vol. 6, No. 2, pp. 33–44, aug 2013.

[4] Munehiro Takimoto. Demand-driven partial dead code elimination. *Information Processing Society of Japan Transactions on Programming*, Vol. 5, No. 1, pp. 9–16, mar 2012.