レジスタ干渉グラフの分割による高速化手法に関する研究

山崎 敬史† 中林 淳一郎† 片岡 正樹† 古関 聰‡ 小松 秀昭‡ 深澤 良彰†
†早稲田大学大学院理工学研究科 ‡日本IBM東京基礎研究所

1 はじめに

レジスタ干渉グラフを用いたレジスタ割り付けは、プログラムを関数ごとにレジスタ干渉グラフを作成している。レジスタ割り付けを行う時、レジスタ数が不足するとレジスタ干渉グラフを作成し直す必要があるが、従来手法では、関数全体に関するレジスタ干渉グラフを作成し直す必要がある。そこで本手法では、関数を複数のブロックに分割し、そのブロックごとにレジスタ干渉グラフを作成する。こうすることで、作成し直すレジスタ干渉グラフのサイズを縮小し、レジスタ割り付けにかかる時間を短縮し、コンパイル速度を上げることができる。

2 従来手法

2.1 グラフ彩色法

レジスタ割り付けにおいて、まず生存区間解析という操作を行う。生存区間解析とは、コンパイラが生成する一時的な変数(仮想レジスタ)の生存区間を調べることである。仮想レジスタ同士の生存区間が重なっている(干渉している)と、その仮想レジスタは異なるレジスタに割り付けられ、生存区間が重なっていなければ、同じレジスタに割り付けることができる[1]。レジスタ干渉グラフを用いたレジスタ割り付けでは、生存区間解析の情報を元に仮想レジスタ同士の干渉情報をグラフにする。そして、このレジスタ干渉グラフを干渉している仮想レジスタ同士は異なる色で彩色していく。こうして実レジスタ数色以下でレジスタ干渉グラフを彩色するという問題を解き、色を実レジスタに対応付けすることで仮想レジスタを実レジスタに割り付けることができる。彩色方法としては、レジスタ干渉グラフをGとし、K(レジスタ数)個未満の干渉を持つ仮想レジスタをMとする。GからMを除去して得られるグラフがK-1色で彩色できればGはK色で彩色可能である。よって、GからK個未満の干渉を持つ仮想レジスタを除去していく。すべての仮想レジスタが除去できれば、仮想レジスタはすべてレジスタに割り付けることができる。しかし、レジスタ数不足により、レジスタ数色以下でレジスタ干渉グラフを彩色できないことがある。そういった時、仮想レジスタをメモリに退避させる作業(スピル)が発生する。スピルが発生すると、仮想レジスタの生存区間情報が変わるので、スピルのたびに仮想レジスタの生存区間を調べ直し、レジスタ干渉グラフを作成し直す必要がある[2]。

2.2 グラフ彩色法の問題点

従来の手法では、関数ごとにレジスタ干渉グラフを作成しているので、スピルが発生した際に作成し直すレジスタ干渉グラフのサイズが大きくなっている。スピルされる仮想レジスタの生存区間が非常に短くても関数全体のレジスタ干渉グラフを作成し直さなければならない。何度もスピルが発生すると、関数全体のレジスタ干渉グラフを作成し直すことはコンパイル速度の低下を引き起こしてしまう。

3 本手法の特徴

本手法では、プログラム中の関数をループ単位で複数のブロックに分割する。ループには多重ループも存在するが、最も外側のループのみに着目し、複数のループブロックを作成する。そして、ループブロックに属さない部分を全てまとめて一つのループ外ブロックとする。本手法では、このブロックごとにレジスタ干渉グラフを作成する。これにより、スピルが発生した際に作り直すレジスタ干渉グラフのサイズが小さくなり、レジスタ割り付けにかかる時間が短くなりコンパイル速度を向上させることができる。

ループブロックを作成した際に、ループブロックの始めから終わりまでブロックを貫いて生存しているにも関

A study on the speedup technique by division of register interference graph
Keiji YAMASAKI †,Junichirou NAKABAYASHI †,Masaki KATAOKA †,Akira KOSEKI ‡,Hideaki KOMATSU ‡,Yoshiaki FUKAZAWA †
† School of Science and Engineering,Waseda University
‡ IBM Japan.Ltd.Tokyo Reserch Laboratory

わらず、ループブロックでは一度も参照されない仮想レジスタ (以後、スルーしている仮想レジスタと呼ぶ) が存在することがある。スルーしている仮想レジスタは、そのループブロックにおいてレジスタに割り付けられている必要はなく、スルーしている仮想レジスタがレジスタに割り付けられることで参照される仮想レジスタがスピルされる可能性が出てくる。よってスルーしている仮想レジスタはレジスタ干渉グラフから除去して、レジスタには割り付けないようにする。こうすることでループブロックの仮想レジスタ数が減少し、レジスタ干渉グラフのサイズを小さくすることができ、よりコンパイル速度を向上させることができる。

4 実験結果

本手法と従来手法を比較するため、バブルソートのプログラムを用いてレジスタ数が4つという環境でレジスタ割り付けを行った。従来手法と本手法でレジスタ割り付けを行った時に作成されるレジスタ干渉グラフを図1と図2に示す。

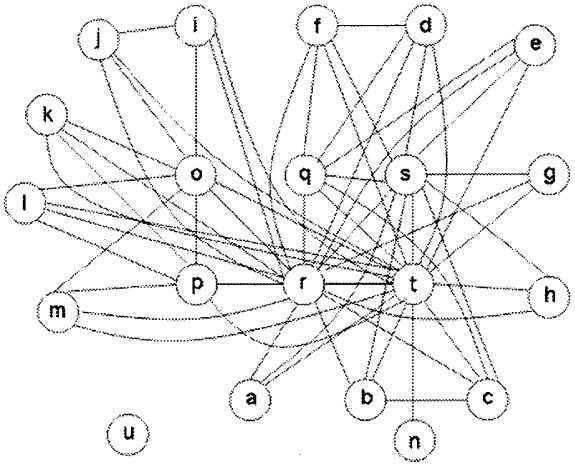

図 1: 従来手法でのレジスタ干渉グラフ

従来手法ではレジスタ干渉グラフが一つ作成されるのに対し、本手法では、図2のようにループ外ブロック、ループブロック1、ループブロック2の3個のブロックに分割された。そして、従来手法では仮想レジスタtとqがスピルされ、本手法ではループブロック1の仮想レジスタkがスピルされた。レジスタ干渉グラフのサイズは各仮想レジスタが干渉している仮想レジスタの数の合計である。従来手法ではまず図1のグラフが作成され、このサイズは122である。次に仮想レジスタtがスピルされ作成されたグラフのサイズが84となり、次に仮想

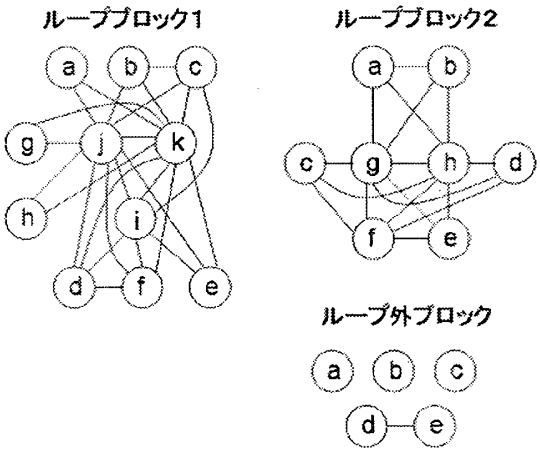

図 2: 本手法でのレジスタ干渉グラフ

レジスタqがスピルされ作成されたグラフのサイズは72となり、合計のサイズは278となった。本手法の各ブロックのサイズは、ループ外ブロックが2、ループブロック1が50、仮想レジスタkがスピルされ作成されたグラフが30、ループブロック2が34となり、合計116となった。よって、従来手法に比べ本手法の方が作成するレジスタ干渉グラフの合計サイズが小さくなるので、レジスタ干渉グラフを作成する時間が短縮され、コンパイル速度が向上することが分かる。

5 まとめ

本稿では、プログラム中の関数をループを基準としたブロックに分割し、そのブロックごとにレジスタ干渉グラフを作成した。またそのブロックで生存しているが一度も参照されない仮想レジスタをレジスタ干渉グラフから取り除いた。これにより、スピルによって作り直されるレジスタ干渉グラフの合計のサイズを小さくし、コンパイル速度を向上させることができた。

参考文献

[1] A.W.Appel "Modern compiler implementation in java"CAMBRIDGE UNIVERSITY PRESS, 1998

[2] G.J.Chaitin "Register allocation and spilling via graph coloring"In Proceedings of the ACM SIGPLAN 1982 Symposium on Compiler Construction, pages 201-207, 1982