A-025

レジスタ割り当てにおけるメモリ転送及びレジスタ間転送の削減手法
A Reduction Method of Data Transfer Instructions in Register Allocation

尾形冬馬 †　　中野秀洋 ‡　　宮内新 ‡
Toma Ogata　Hidehiro Nakano　Arata Miyauchi

1. はじめに

近年プロセッサの高速化が進む一方，プロセッサとキャッシュメモリ(メモリ)の動作速度のギャップは未だ大きいままである．そこで最も高速にアクセス可能な記憶領域であるレジスタを有効活用する事が重要となる．

2. 従来手法

従来のレジスタ割り当て手法として，グラフ彩色を用いた Graph Coloring Register Allocation (GCRA)[1]及びこれを改良した SSA-Based Register Allocation (SSA-RA) [2]がある．

GCRA では各変数が同時に生存するかどうかを表す干渉グラフを作成し，これに対しグラフの頂点彩色を行う事で割り当てを行う．しかし GCRA では変数の生存区間に対して一括して割り当てを行うため，無駄なスピルが発生するという問題点がある．

SSA-RA では，グラフ彩色での割り当てを行う前に変数の生存区間の分割を行う事で無駄なスピルを最少に抑えられる．しかし SSA-RA では事前に変数を分割した結果として，レジスタ間の転送命令が増加するという問題点が存在する．そこで無駄なスピルを最小限にしつつ，レジスタ間の転送命令の増加を抑える手法を提案する．

3. 提案手法

本手法では以下に示すレジスタ割り当ての十分条件に注目する．

割り当ての十分条件
全ての命令実行の時点において生存する変数の数がレジスタ数以下である．

十分条件を満たす変数群は，一部の変数を除きレジスタを割り当てる事が可能であり，また割り当て不可能な変数もレジスタ間転送命令を追加する事で割り当てる事が可能である．

3.1 Pier Bridge Table

本手法では図 1 に示すような Pier-Bridge Table (PBT) と呼ぶ表を用いて割り当てを行う．

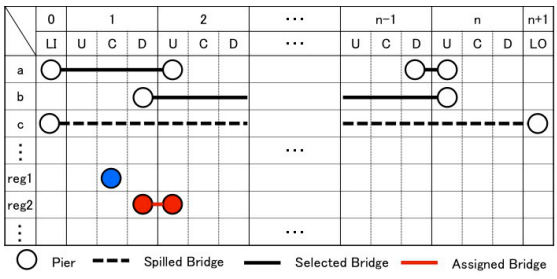

図 1　Pier – Bridge Table

PBT は基本ブロック毎に作成される．PBT の列は命令列，行は変数及びレジスタを表す．また，各列には 3 つのフィールドがあり，それぞれ変数の定義，上書き，使用に対応する．なお LI，LO フィールドは基本ブロックを跨いで生存する変数を表す．

変数が定義，上書き，使用される場所に対応するセルに Pier が作られ，Pier 間の変数が生存する区間に Bridge が作られる．

さらに，Bridge は以下に示す 3 状態に分類される．
Spilled：メモリへスピルされた状態
Selected：Bridge Select にて選択された状態
Assigned：レジスタが割り当てられた状態

本手法では Bridge の状態を順に変更する事で割り当てを行う．本手法の割り当て手順を以下に示す．

図 2　提案手法の手順

以降で各手順の詳細を説明する．

3.1 Build

プログラムを先頭から走査する事で PBT を作成する．生成時全ての Bridge は Spilled 状態とする．

3.2 Bridge Select

全てのフィールドにおいて生存している変数の数がレジスタ数以下になるよう Bridge を選択し Selected 状態に変更する．図 3 に Bridge Select のフローチャートを示す．

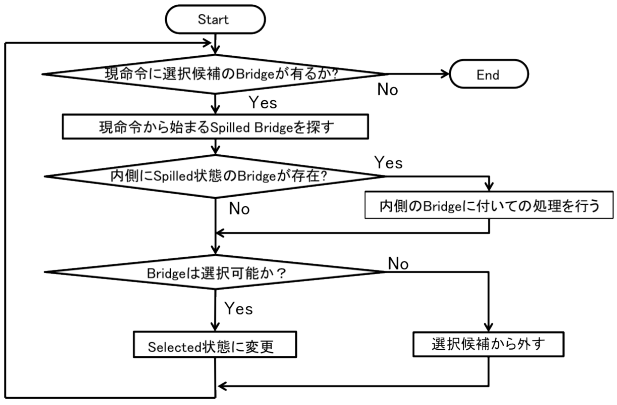

図 3 Bridge Select のフローチャート

ここで，Bridge が選択可能であるとは，その Bridge を Selected 状態に変えたとしても以下の条件を満たす場合である．

PBT に対する割り当ての十分条件
全てのフィールドにおいて Pier の数と Spilled 状態を

† 東京都市大学大学院 工学研究科
‡ 東京都市大学 知識工学部 情報科学科

除く Bridge の数の和がレジスタ数以下である.
PBT の先頭の命令から順に図 3 に示すフローチャートに従い Bridge の状態を変更する.

3.3 Joint Blocks

LI, LO フィールドに架かる Bridge の選択を行う.
LI に架かる Bridge は対応する LO Bridge が Selected 状態である場合のみ, Selected 状態に変更を行う. 具体的には, 図 4 に示すフローチャートに基づいて LI, LO Bridge の選択を行う.

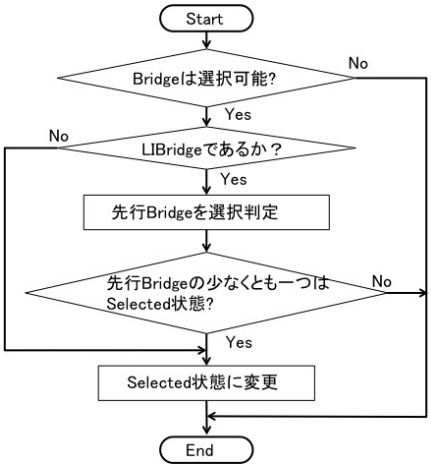

図 4 Joint Block のフローチャート

3.4 Allocation

Selected Bridge に対し先頭から順にレジスタの割り当てを行う. 一つのレジスタに割り当てが不可能である Bridge が発生した場合, 適宜レジスタ間転送命令を挿入する事で割り当てが可能となる.
また, 対応する LI, LO Bridge に異なるレジスタが割り当てられる事を防ぐため, Bridge に複数の割り当て候補レジスタが存在する場合には, 既に割り当てられている Bridge を優先して割り当てるレジスタを決定する.

3.5 Apply

Allocation を行った PBT の結果を元にメモリ転送命令及びレジスタ間転送命令を追加する. Spilled Bridge が存在する場合, Store 及び Load 命令を追加する必要がある. また, 対応する LI, LO Bridge の状態が異なる場合, 適宜 Load, Store, レジスタ間転送命令を追加する.

4. 評価

本手法の有効性を確かめるため従来の手法との比較実験を行った. 比較の対象は RCA 及び SSA-RA とした. COINS[3]のレジスタ割り当て部に各割り当て方式を実装し, 各ベンチマークをコンパイル, 実行しクロック数の測定を行った. 性能の測定には Linux の perf performance counter を使用した. 各測定は 50 回試行し, その平均値を評価する.
ベンチマークには Dhrystone, Dijkstra, Secure Hash Algorithm を用いた. また各ベンチマークは -O0,-O1,-O2,-O3 の 4 種類の最適化を用いてコンパイルを行い, 計 12 個のプログラムで評価を行った.

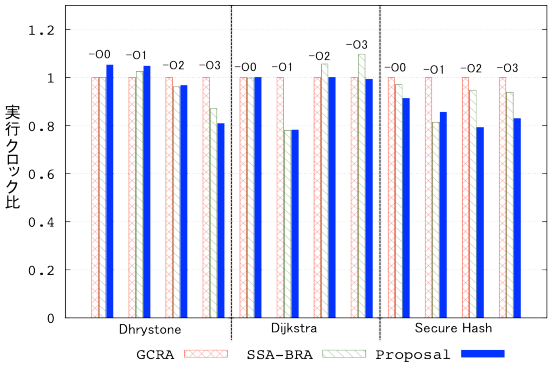

図 5 実行結果:クロック数

図 5 に各ベンチマークの実行クロック比を示す. この値は各ベンチマークにおける GCRA の実行クロック数を基準としている.
従来の手法と比べ大半のベンチマークにおいて本手法は優れており, GCRA と比べ平均して 8%, SSA-RA と比べ 3%クロック数が減少している. 特に本手法は最適化を行ったベンチマークに対して用いた場合, GCRA と比べ 10%, SSA-RA と比べ 5%クロック数が減少している. これは, 最適化を行う事で変数の生存区間がより複雑になるため, 従来の手法では無駄な Spill や転送命令が追加されてしまう一方, 本手法では必要な場合のみ転送命令を追加する事でこれらを回避している為であると考えられる.

5. 結論

本稿ではメモリアクセスの回数を減らしつつレジスタ間の転送命令の増加を抑制するレジスタ割り当て手法を提案した. 従来の割り当て手法ではグラフ彩色を利用するため, 割り当ての実行中に変数を分割する事が困難であった. そこで我々はグラフ彩色を用いない新たな割り当て手法を提案した. ベンチマークを用いて実験を行った結果, 本手法は従来の手法と比べクロック数を削減する事ができ, また最適化を行ったベンチマークに対してはより良い結果となった.
今後の課題として, Bridge Select 及び Joint Blocks の選択アルゴリズムの改良や, より多くのベンチマークに対する評価などが挙げられる.

6. 参考文献

[1]Gregory J. Chaitin, et al Register allocation via coloring", Computer Languages, Vol 6, pp.47-57, (1981)
[2]Sebastian Hack, et al："Register allocation for programs in SSA-form", Compiler Construction 2006, pages 247?262, Springer, (2006)
[3]中田 育男 他："コンパイラの基盤技術と実践", 朝倉書店, 2008.