シングルチップ・マイコン用コンパイラのレジスタ割り当て手法

5D−6

磯崎 博子 、 木村 隆子 、 小野 洋彦 、 黒田 一朗
日本電気(株)

1 はじめに

汎用レジスタを持つRISC/CISCマイクロコンピュータ用のコンパイラでは、レジスタ割り当て手法としてグラフ彩色法等の効率の良い手法が確立されている。しかし、民生等組み込み用に使用する8/16ビットシングルチップ・マイコンは、低コスト重視の観点からレジスタ本数を抑え、アキュムレータ方式を採用している場合が多い。そこで、限られたレジスタを効率的に使用するためのレジスタ割り当て手法として提案されているレジスタトラッキング手法[1]が有効だと考えられる。

本稿では、このレジスタトラッキング手法に、レジスタに準ずるコストを持つ高速メモリを準レジスタとしてリソースに加える手法を提案し、効率的にレジスタを使用できることを性能評価に基づいて述べる。

2 ターゲットアーキテクチャ

ターゲットである78K0マイコンは以下の特徴を持つ。

- ●8ビットレジスタ8本(A,X,B,C,D,E,H,L)×4Bank
  2本の8ビットレジスタをペアとして使用可能
- ●アキュームレータ方式の演算
  A,AXでのみの演算,Load/Storeのレジスタの制限
- ● 高速内部RAM領域 (以降"高速メモリ")
  レジスタに準ずるコストを持つ

シングルチップ・マイコンでは、生成オブジェクトの実行速度以上にオブジェクトサイズが重視される。そこで本稿では省サイズを中心に述べ、コスト計算は命令バイト数を基本とする。各命令に要するバイト数を表1に示す。

		8ビット	16ビット
転送	A ← r (レジスタ)	1	1
	A ← 高速メモリ	2	2
	A ← メモリ	3	3
演算	A op r (レジスタ)	2	6
	A op 高速メモリ	2	6
	A op メモリ	3	8

表1 命令バイト数

A Register Allocation for a Singlechip Microcomputer.
Hiroko Isozaki , Takako Kimura , Hirohiko Ono , Ichiro Kuroda
NEC Co.

3 レジスタトラッキング

レジスタトラッキングの概要を説明する。
まず、生変数解析により各変数のLive/Dead情報を収集する。次にコード生成部では、register association listを作成する。これは各レジスタにつき格納される変数/テンポラリのリストで、Live/Dead情報、メモリへのセーブの必要の有無を示す情報を持ち、コード生成をしながら更新していく。コード生成の過程でレジスタが必要になった時は、レジスタフリー関数が呼ばれる。この関数ではregister association listの各レジスタの状態からコスト計算式に基づきコスト最小のレジスタを選択する。また、演算や代入の命令を生成する際には、オペランドの再利用をするために、オペランドの変数/テンポラリを一旦レジスタに載せた後に演算をするべきか否かをコスト計算を行なって決定する。

4 従来の問題点

レジスタトラッキングではレジスタの不足によるスタックの消費が問題となる。レジスタが全て使用状態になると、スタック領域を使用せざるを得ない。ところがスタックはアクセスコストが高いので、結局レジスタには蓄えずに従来通りメモリからアクセスする場合が多い。78K0では16ビット変数4つを割り当てると、全てのレジスタが使用状態になってしまうことから、この状況の発生率は高い。

5 レジスタトラッキングでの高速メモリ活用

78K0マイコンでは表1に示す通り、転送コストはレジスタに劣るが、演算コストはレジスタと等しい高速メモリを備えている。そこで、高速メモリをレジスタに準ずるものとして活用し、レジスタの不足を解消する手法を提案する。従来の手法とは以下の2点が異なる。

- ●高速メモリを退避先の候補として追加
- ●コスト計算に現時点以降の参照時のコストを追加
  コスト計算をより正確にするため、生変数解析時に参照回数を求め、退避したリソースによる参照時のコストを追加した。

5.1 コスト計算式

78K0の高速メモリは転送コストがレジスタよりも高いので、以降の参照回数によって総コストが変動する。

そこで、従来はレジスタのロード/ストアに要するコストを求めてコスト関数としていたのに対し、本手法では各変数/テンポラリの参照回数を求めておき、参照時のコストも加算するように改善した。

コスト＝変更コスト＋参照コスト

変更コスト＝

レジスタ配置処理(退避等)に要するバイト数

参照コスト＝

演算の実行に要するバイト数×演算参照回数

＋代入式の実行に要するバイト数×代入参照回数

例として8ビットのデータが格納される場所毎の参照コストを表2に示す。

格納場所	参照コスト計算式
メモリ	3×n＋3×nt
高速メモリ	2×n＋2×nt
dead レジスタ	2×n＋1×nt
live レジスタ	2×n＋1×nt＋(2-2)×n'＋(2-1)×nt'

n: 演算参照回数, nt: 代入参照回数

n'/nt':live レジスタに格納されていた変数の演算/代入参照回数

表2　8ビットデータの参照コスト

5.2　レジスタフリー関数

アキュムレータレジスタAをフリーの状態にするレジスタフリー関数getregA()で使用するコスト計算式のうち、変更コストの計算方法は従来通りで、退避先候補に高速メモリを追加したのみである。表3に8ビットの場合を示す。

Aの状態	退避先	cost	退避命令
Dead	退避は不要	0	
Live	メモリ	0	
&	高速メモリ	2	A→高速メモリ
Not to be	dead レジスタ	1	A→r
Saved	live レジスタ	3	A←→r, A→高速メモリ
Live	メモリ	3	A→メモリ
&	高速メモリ	2	A→高速メモリ
to be	dead レジスタ	1	A→r
Saved	live レジスタ	3	A←→r, A→高速メモリ

表3 getregA()の変更コスト

getregA()では「5.1 コスト計算式」に従い、上記の変更コストにAの退避先の参照コスト(表2参照)を加算したものをコストとして、その中で最小のコストを持つ配置方法を選択する。

5.3　オペランドの再利用

コード生成の過程で、オペランドの再利用を考慮して、オペランドをどこに配置すれば良いかのコスト計算を行なう。ここでは従来の変更コストに演算のコストを含める。これはデータの移動により演算処理のコストが異なってくるためである。例として表4に8ビットのAレジスタと変数(onメモリ)の演算の場合の変更コストを示す。

配置先	cost	配置命令＋演算命令
メモリ	3	ALU A, メモリ
高速メモリ	11	メモリ→高速メモリ, ALU A, 高速メモリ
レジスタ	10	メモリ→r, ALU A, r

表4 A op mem の変更コスト

コスト計算部は「5.1 コスト計算式」に従い、上記の変更コストにメモリの配置先の参照コスト(表2参照)を加算したものをコストとし、その中で最小のコストを持つ配置方法を選択する。

6　評価

従来のレジスタトラッキング手法と、本稿の高速メモリを準レジスタとして活用するレジスタトラッキング手法について、データ処理に関するサンプルプログラムを評価した結果を表5に示す。

サイズ	従来方式	本方式	増減
コード	282	264	-18 (6.4% 減)
データ	36	36	
高速メモリ	0	4	+4
合計	318	304	-16 (4.4% 減)

表5 サンプルプログラムでの評価 (単位：バイト)

7　考察／結論

本稿で述べた高速メモリを準レジスタとして活用するレジスタトラッキング手法は、基本ブロックが長く、使用する変数が多いデータ処理中心のサンプルプログラムに対し、コードサイズに加え総ROM+RAMサイズにおいても減少が認められ、性能向上が確認された。

組み込み用途のプログラムの特徴として、データ処理中心のプログラムと制御系プログラムに大別される。データ処理中心プログラムでは前述の通り本手法の有効性が認められたが、制御系プログラムは条件判断が多く基本ブロックが細かく分断されるため、基本ブロックを越えてコスト計算、レジスタ割り当てを行なうレジスタトラッキングを行なうことにより性能向上を図ることが可能と推測される。

8　課題

今後は、組み込み用途の制御系プログラムに対しても効率の良いコンパイラを開発するために、基本ブロックを越えたレジスタトラッキング手法について検討する予定である。

参考文献

[1] CRAFTING A COMPILER WITH C. Fischer, LeBlanc, 1991