VLIW型計算機KIDOCH用Cコンパイラについて

6X-4

永田 仁史　　安倍 正人　　城戸 健一

(東北大学応用情報学研究センター)

1. はじめに

KIDOCHは音響信号処理用に開発された高速演算装置で、128bitの水平型マイクロ命令を持つVLIW型計算機である。この計算機を動かすために、 C言語をマイクロコードに変換するコンパイラを開発した。コンパイラの設計においてはコンパクションによる目的コードの最適化が一つの大きな問題点であったが、これについては既に報告した*sup*1)。今回はコンパクションの前処理としての中間コードにおける最適化、およびハードウェア構成を生かすようなメモリ管理によって関数呼び出しを効率化した点などについて述べる。

2. コンパイラの構成

C言語をコンパイルした結果として最終的に出力するものは 128bitのマイクロコードであり、そこへ至るまでの流れを図1に示す。ソースプログラムのKIDOCH-C言語はふつうのCと異る点として次のような点が挙げられる。

(1) 変数の宣言部で、2つあるメモリユニットのどちらを使うかを指定できる。

(2) レジスタ変数の宣言部で、2つあるALUのどちらに附属するレジスタを使うかを指定きる。

(3) 複素数型のデータが使用できる。

このようなC言語をまずYACCによって構文解析し、 4つ組による中間コードの段階で、ある程度の最適化とコンパクション部とのマッチングを行う。この結果出力される4つ組を入力とし、演算器動作表を使ってマイクロコードと一対一に対応するKIDOCHアセンブラを生成する。そしてアセンブラの段階でリンク及び番地付けを行い、アセンブルしてマイクロコードを生成する。

3. 中間コードにおける最適化

最適化については最適化プログラムの作成に法外な努力をしなくても可能な改良のうちの大部分を行うことができるもののみを取り入れる方針であり、ここでは図1に示すようなことを行っている。ここで、ループの変換はコンパクションに関係した最適化であり、これについて説明する。

3.1. ループの変換

Cではいくつかのループ文があるが、図2、3のfor文、while文では、中間コードの並びにおいて条件判断とループ内の文の間にjump命令が入り、ブロックが区切られる。コンパクションは基本ブロックごとに行うようになっており、このとき、基本ブロックが大きい程、そのブロックの最適化がうまく行くという傾向がある。そこで図2、3、(2)のように条件判断のブロックをループの終りにコピーし、 do-while型の文に書き直すと条件判断とループのブロックがつながり、大きなブロックとなる。条件判断の式が簡単なものならば、この変換によって条件判断だけ単独でコンパクションした場合のステップ数だけ少なくなる。尚、更にループの形を変形してコンパクションを効果的にするループスケジューリング法も検討中である。

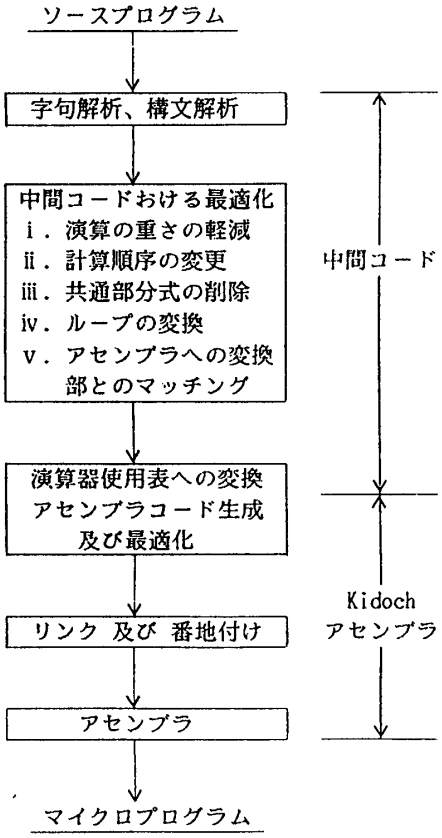


図1　コンパイラの構成

The C Compiler of VLIW Type Computer KIDOCH

Yoshifumi Nagata, Masato Abe, and Ken'iti Kido

Research Center for Applied Information Sciences, Tohoku University

4. アセンブラへの変換部

ここではテーブル駆動型のコード生成法によってコンパクションを行っている。生成されるアセンブラコードの効率は2節で述べたように変数宣言の際のユニット指定によって変わるので、いくつか試してから効率の良いものを選ぶことができる。このようなソースプログラムの変更によって最適化が図れるように、次のような情報をコンパイルの際に返すようにしている。

(1) ALU使用頻度:

一方のALUの使用頻度に偏りがある場合はそのALUのレジスタ変数を減らす。またはもう一方のレジスタ変数を増やす。

(2) 計算の中間結果のDMUへの一時退避頻度:

多い場合はレジスタ変数を減らす。

(3) DMU使用頻度:

一方のDMUの使用頻度に偏りがある場合は変数のDMUユニットを変更する。

(4) ステップ数:

以上の変更による効果を確かめる。

コンパクションの際、発生した中間結果を後で使う場所へ転送できずに行き詰まってしまう場合がある。これはそのデータの発生と使用が離れている場合等に起り易い。このようなときはデータの転送経路を確保するような中間コードを挿入してからもう一回コンパクションをやり直す。行き詰まったときの状況を判断して次のいずれかを行った後、ブロックの最初に戻る。

(1) 発生したデータの後にそのデータのレジスタへの格納命令を挿入する。

(2) 発生したデータの後にそのデータのメモリへの格納命令を挿入する。

(3) そのデータを発生させた中間コードをそのステップに埋め込むことを禁止する。

5. 関数呼び出しとメモリの管理

ふつうのC言語と同様に呼び出しの制御情報を置くスタックと局所変数を置くフレームメモリを用いて関数呼び出しを行っている。そしてその制御のためにレジスタによるポインタを2つ用いている。ポインタ用のレジスタのあるALUは使用頻度が高くなるので、これを分散するためにスタックポインタはALU1に、フレームポインタはALU2に置くことにした。

またレジスタファイルが無いので、関数呼び出しの際にレジスタの退避を行わなければならない。これが引数の退避と重なるとメモリへのアクセスが連続し、効率が悪くなる。

そこで、引数の領域とフレームメモリを一緒にしてDMU2に置き、 DMU1には引数以外の制御情報とレジスタの退避した値を置くことにした。この結果、引数の退避はDMU2へALU2によるアドレス計算によって行い、レジスタ退避とその他の制御情報の退避はDMU1へALU1のアドレス計算によって行うことになり、作業が分散されて効率が良くなる。しかし2つのメモリを関数呼び出し用に使ってしまうと、プログラム内でメモリバンクを切り換えて使うとができなくなる。そこでフレームメモリおよび引数の領域をDMU1だけに置くようにも切り換えられるようになっており、場合に応じて使い分けられる。実効時のメモリ内容を図4に示す。

6. まとめ

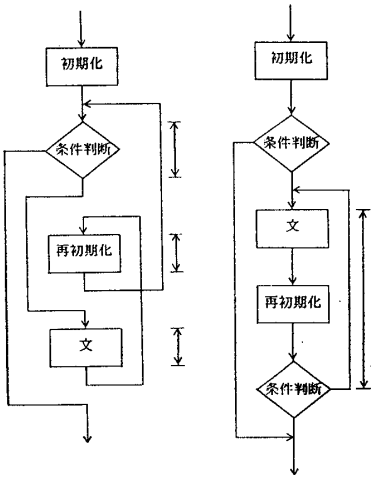
KIDOCH-Cコンパイラの特徴をまとめると次のようになる。

(1) 高級言語からVLIW型のマイクロコードへの変換を行うコンパイラであり、マイクロコードへの変換をテーブル駆動型のコード生成法を用いて行っている。

(2) 2つのALUとDMUを用いて、関数呼び出し時の退避作業を分散し、効率化を図っている。

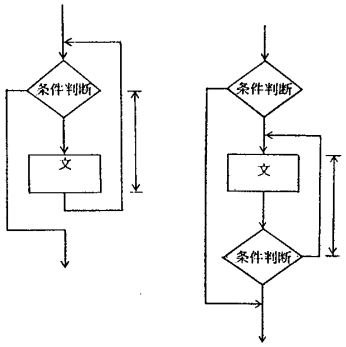
参考文献

[1] 永田、他:VLIW型計算機KIDOCH用Cコンパイラ、第15回東北大学応用情報学研究センターシンポジウム予稿集(1989)。



(a)書き換え前 (b)書き換え後

図2 for文



(a)書き換え前 (b)書き換え後

図3 while文

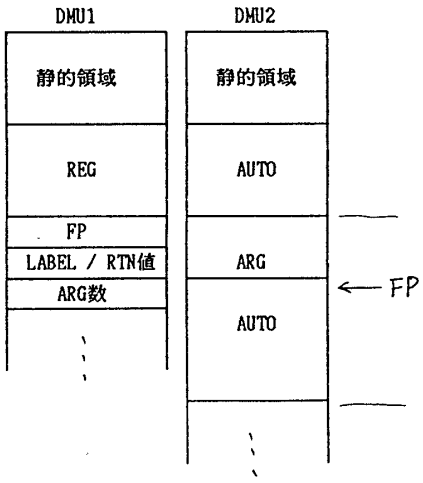


図4 実行時のメモリ内容