GCC を用いた ASIP 用リターゲッタブルコンパイラジェネレータの開発

Development of a retargetable compiler generator for ASIPs using gcc

引地 信之 [1)]　　今井 正治 [2)]　　武内 良典 [3)]
Nobuyuki HICHI　Masaharu IMAI　Yoshinori TAKEUCHI

1) (株) ＳＲＡ先端技術研究所　(〒160-0004 東京都新宿区四谷 3-12 丸正ビル 5F
Email: hikichi@sra.co.jp)
2) 大阪大学 大学院 情報科学研究科 (〒560-8531 大阪府 豊中市 待兼山町 1-3
Email: imai@ist.osaka-u.ac.jp)
3) 大阪大学 大学院 情報科学研究科 (〒560-8531 大阪府 豊中市 待兼山町 1-3
Email: takeuchi@ist.osaka-u.ac.jp)

ABSTRACT. In this paper, a prototype compiler generator based on GCC for hardware architect is discussed. The proposed compiler generator uses architecture information of PEAS-III system, and produces header file (tm.h) and machine description (md) for retarget of GCC. Experimental results show that the compilers for MIPS-R3000 and its derivatives were generated, and optimization feature of GCC has been enabled in this compiler generator.

１．背景

デジタル信号処理などの特定用途向けシステムを構築する場合、いくつかの方法がある。一つは DSP を用いて実現する方法、汎用プロセッサと専用の ASIC (Application Specific Integrated Circuit) を併用する方法、ASIP (Application Specific Instruction-set Processors) を用いて実現する方法などがある。DSP を用いる方法では、低コストで比較的性能のよいシステムを構築できるが、DSP の特長を生かすためには、ボトルネックとなる部分をアセンブラでプログラミングする必要がある。そのため，開発工数がかかり，さらにその部分の移植性がなくなってしまう問題点がある。

汎用プロセッサと専用の ASIC を併用する方法では、出荷個数によっては量産効果が期待でき、非常に低コストとなるが、ボトルネックのプログラミングでは，性能を出すために ASIC を並列に動作させるプログラミングが必要になる。並列プログラミングは, 一般には DSP を用いたプログラミングよりも困難であるため，開発効率は低下する。さらに、アセンブリプログラミングを行うために移植性も低下する。

ASIP を用いる方法では、アプリケーションにあわせたハードウェア構成と命令セットのチューニングが可能となり、ハードウェアコストや消費電力に対して最適な設計となる潜在的な能力を備えている。ASIP の開発システムが整備されていれば、アプリケーション･プログラムの移植性も兼ね備えたものにできる。

PEAS-III(Practical Environment for ASIP development)システム[1] は、効率よく ASIP を設計するためのハードウェア／ソフトウェア・コデザイン・システムである。PEAS-III システムでは、プロセッサの設計をアーキテクチャ記述により指定し、ハードウェア合成可能な HDL 記述を生成する方式を採用している。アーキテクチャ記述は、(1) パイプラインステージ数や分岐遅延などを含むアーキテクチャパラメータ、(2) プロセッサで使用するリソース、(3) 命令フォーマット、(4) マイクロ動作記述(クロック単位の命令の動作記述、命令ごとに指定)、(5) 割り込みの定義、の 5 種類の記述からなる。PEAS-III システム向けの開発環境のうち、コンパイラ向けのコンパイラ生成系は試作評価されているところである[2, 3]。文献[2, 3]では、コンパイラ生成系の入力として 5 種類のアーキテクチャ情報を策定し、既存の商用コンパイラ開発環境と組み合わせることにより自動生成できることを確認していたが、本論文では広く用いられている GNU C コンパイラを用いたコンパイラ生成系の実装アプローチとその結果について紹介する。

２．目的

PEAS-III システムではコンパイラをはじめとするソフトウェア開発環境が重要な働きをする。大学などで利用する場合、あるいは一般に評価する場合には、商用コンパイラ開発環境が必要となるコンパイラ生成系の採用は困難である。フリーソフトウェアである開発環境向けのコンパイラ生成系があれば、PEAS-III システムを教育目的でも採用しやすくなる。

そこで、PEAS-III システム向けのコンパイラ生成系をフリーソフトウェアである GNU C コンパイラをベースとして試作したのが、本システムである。

これにより、利用者のニーズによってコンパイラ生成系を使い分けることが可能となる。コンパイラ生成系の入力となるアーキテクチャ情報は、バックエンドとして利用するコンパイラに依存しないように、仕様を策定した。

二つ以外のコンパイラにも対応しうる仕様であると考えており、さらにアーキテクチャ情報の標準化まで進めていくことが可能となる。これにより、組込みプロセッサ用のコンパイラ生成技術およびプロセッサおよび応用プログラムの最適化技術の研究開発に貢献できると考えてられる。

３．コンパイラ生成系の概要

（1）アーキテクチャ情報

コンパイラ生成系の入力記述としてアーキテクチャ情報を、次のような 5 項目により指定する。[2, 3]

・リソース情報
　ターゲット・アーキテクチャを構成するリソースインスタンスの仕様情報をクラス定義として格納する。
・リソース・インスタンス間接続情報
　リソース情報で与えられたクラスからのインスタンスとして資源を定義しつつ、各資源の接続情報も与える。
・ストレージ情報
　レジスタ、レジスタファイル、メモリなどの記憶素子についての情報、スタックの仕様などを記述する。
・ファンクション情報
　機能の定義
　内部動作に関わる機能
　　ストレージ情報で定義されたリソースを用いる。
　外部動作に関わる機能
　　記憶用以外を対象とした機能、算術論理演算動を扱う。
　制御用の機能
・命令セット情報
　命令ごとに次の定義を行う。
　- オペランド
　- 命令フォーマット
　- 各ステージで使用するリソース
　- 命令のビヘイビア

例を左に示す。

```
命令セット情報の例
SUBU
  {
    operand
    {
      GPR UInt32to0 a;
      GPR UInt32to0 b;
      GPR UInt32to0 c;
    }
    format
    {
      "SUBU" a ", " b ", " c
    }
    functions
    {
      stage(1)
      {
        PC.direct_read
        IMEM.load_word
        PC.inc
        IR.direct_read
      }
      stage(2)
      {
        GPR.read0(b)
        GPR.read1(c)
      }
      stage(3)
      {
        ALU0.subtract_u
      }
      stage(4)
      {}
      stage(5)
      {
        GPR.write(a)
      }
    }
    behavior
    {
      a = b - c;
```

（2）コンパイラ生成系の処理手順

コンパイラ生成系は、アーキテクチャ情報を入力として、GCC のリターゲットに必要な、tm.h と md ファイル[4]を生成することで実現する。md ファイルは、GCC 内の仮想命令(半固定の命令でエントリ名がついているものとついていないものがある) は、ターゲット　CPU のどのような命令あるいは命令列に対応するかを記述している。tm.h ではターゲット CPU の語長やレジスタの用法などを定義する。R3000 準拠のアーキテクチャをベースプロセッサ・アーキテクチャとして選択しているため、現在のコンパイラ生成系では tm.h では適切な処理の規定値を選ぶような戦略を採用し、md ファイルのみを生成する構成としている。

GCC では md ファイル内で、そのアーキテクチャ特有な最適化をいろいろ指定できる。ターゲット・アーキテクチャ固有の最適化戦略をアーキテクチャ情報から自動合成する方法は今後の課題である。

md ファイル内のエントリの生成方法には、次のような種類がある。

a）単純な単一命令のマッピング
b）単一命令のマッピングであるがオペランドを変化させたもの
c）推移規則などを適用して複数の命令へのマッピング
d）テンプレートを用意して、それに沿ったマッピングを行うもの

最初の 3 種類に関して、具体的な例を次に示す。

32 ビット減算の場合は、単純なマッピングにより生成させることができる。生成されたエントリを次に示す。

```
(define_insn "subsi3"
  [(set (match_operand:SI 0 "register_operand" "=r")
        (minus:SI (match_operand:SI 1 "register_operand" "r")
                  (match_operand:SI 2 "register_operand" "r")))]
  ""
  "*
{
  switch (which_alternative)
    {
    case 0:
        return ¥"subu¥¥t%0,%1,%2¥";
    }
}"
```

このエントリの意味している点の概略を次に示す。

subsi3 という 3 引数の Simode(32 ビットの基本データタイプでターゲットに依存しない GCC 内部の固有のデータ)のエントリは次のようにマッピングする。

マッピングの前提条件としては、適合した繊部ランド 3 つがレジスタにあれば、このマッチング規則を適用する。Match_operand:SI 数字 “register_operand” により前提条件を記述している。

subsi3 という意味は 第一引数から 第二< 引数を引いて(minus:SI) それを第一引数に代入する(set)となる。

前庭条件を満たせばマッチング規則を適用し、コード生成する際には、which_alternative(選択肢がある場合何番目の選択肢かを保持する変数) の値によって、コード生成する命令を切り替えている。前期の例では選択肢がないので、生成する命令も一種類だけとなっている。

コンパイラ生成系は switch 文の外側のブロックを自動的に決定している。

二の補数をとるエントリ negsi2 や、一の補数をとる one_cmplsi2 というエントリがある。R3000 準拠の命令セットでは、直接の命令がないので、直接対応する命令がない場合、ゼロレジスタ(次の例では $0で表現) とほかの演算を利用したものにマッピングするような規則を導入した。これにより生成された md のエントリを次に示す。

```
(define_insn "negsi2"
  [(set (match_operand:SI 0 "register_operand" "=r")
        (neg:SI (match_operand:SI 1 "register_operand" "r")))]
  ""
  "*
{
  switch (which_alternative)
    {
    case 0:
         return ¥"subu¥¥t%0,$0,%1¥";
    }
}"
```

32 ビットの除算のエントリ名は、divsi3 である。R3000 準拠の命令セットでは、直接除算を行って汎用レジスタに結果を得るような命令がないので、前述の (3) の方針に従って、複数の命令を利用して汎用レジスタから結果をえられるようなものを検索するような規則を導入し、生成するようにした。

```
(define_insn "divsi3"
  [(set (match_operand:SI 0 "register_operand" "=r")
        (div:SI (match_operand:SI 1 "register_operand" "r")
                (match_operand:SI 2 "register_operand" "r")))]
  ""
  "*
   output_asm_insn (¥"div¥¥t%1,%2¥", operands);
   return ¥"mflo¥¥t%0¥";
")
```

４．評価実験

(1) 実験の目的

実験の目的は，次の項目を確認することにある。

i) 商用ではないフリーソフトウェアである開発環境向のコンパイラ生成系の構築。
ii) 生成されたコンパイラがアプリケーション･プログラムに対して期待する動作を行うようなコード生成を行うこと。
iii) ターゲット CPU アーキテクチャを変化させてコンパイラを生成した場合、得られたコンパイラが、変更されたアーキテクチャに対応したコード生成を行うこと。
vi) 生成コードの最適化の適用。

(2) ベースプロセッサ・アーキテクチャ

特定のアプリケーション向けにアーキテクチャをカスタマイズしていない CPU アーキテクチャを、ベースプロセッサ・アーキテクチャと呼ぶことにする。今回の評価では、次のようなアーキテクチャを仮定した。

命令セットアーキテクチャ
32 ビット RISC プロセッサ R3000 のサブセット、浮動小数点数関連の命令を含むコプロセッサ関連の命令はない。

命令の実行クロック数
すべて 5 クロックで終了するようなパイプライン構成。

(3) アプリケーション

評価用アプリケーション･プログラムとして、離散コサイン変換(DCT)と FIR フィルタ(整数型)、複素係数 FIR フィルタ(固定小数点数型)を使用した。プログラムの特徴として、FIR フィルタでは、もっとも実行回数の多いループの本体で定数との乗算と加算が行われている。DCT では､定数との乗算と定数とのシフトの演算が多い。複素数 FIR フィルタでは、変数の乗算と定数とのシフトの演算が多い。そこでベースプロセッサ・アーキテクチャに、乗算とシフトを行う命令(MSRA、 Multiply and Shift Right Arithmetic)を追加したアーキテクチャを想定し(MSRA 命令も 5 クロックで終了)､アプリケーションの実行クロック数の変化を調べた。

(4) 評価実験環境

本実験では MIPS R3000 のシンボリック命令シミュレータである spim[6] をベースに改造して(GCC[4, 5] からの生成コードとのインタフェースをとるための修正も含む)、実行サイクル数(便宜上、5 クロックを 1 サイクルとして扱う)を計測できるように改良したシミュレータを用いた｡spim はシンボリックシミュレータであるため、アセンブラの機能も含まれている点も有効に活用することが可能であった。

(5) 実験結果

ベースプロセッサ・アーキテクチャと乗算シフト命令 MSRA を追加したアーキテクチャを想定し､それぞれ向けのアーキテクチャ情報を作成し、それぞれのアーキテクチャ向けのクロスコンパイラを GCC への設定ファイルを作成する方法 (ターゲット・アーキテクチャ固有の最適化を無効にするためにバージョン 1.42 を用いた) を用いて作成した。評価用アプリケーション･プログラムに対して、クロスコンパイラを用いてコード生成を行い、評価用に改造した spim により、実行サイクル数を計測した結果を、次の表に示す。なお、ベースプロセッサ・アーキテクチャ向けの GCC として見た場合、固有の最適化記述は行っていないため、ターゲット・アーキテクチャに依存しない GCC の最適化の度合いを調べるために、最適化オプション(-O)を指定しない場合と、指定した場合をものせた。

ベースプロセッサ・アーキテクチャ　　単位: サイクル

アプリケーション名	-O 与えない場合	-O を与えた場合
DCT	388	250
複素係数 FIR フィルタ	10057	5027
FIR フィルタ	809	503

MSRA を追加したアーキテクチャ　　単位: サイクル

アプリケーション名	-O 与えない場合	-O を与えた場合
DCT	388	233
複素係数 FIR フィルタ	10057	4627
FIR フィルタ	809	503

以上の表から次の知見が得られた。

・ベースプロセッサ・アーキテクチャの場合
最適化オプションを指定すると実行速度が 36% から 50％ まで(平均で 41%)削減可能であることが知られた。もっとも最適化が有効だったものは複素係数 FIR フィルタであった。

・MSRA を追加したアーキテクチャの場合
傾向は ベースプロセッサ･アーキテクチャの場合と同じである。最適化オプションを指定すると実行速度が 38% から 54％ まで(平均で 44%)削減可能であることがわかる。
MSRA 命令の追加は､最適化オプションを追加した、DCT と 複素係数 FIR で有効であり、この命令の追加により実行サイクル数が 7~8％ 削除可能となった。
複素係数 FIR フィルタの場合の、最適化オプシ

ョンを与えたとき、MSRA 命令がないアーキテクチャと、ある場合のアーキテクチャの生成コードを比較して眺めると、コード生成されたアセンブリファイルの行数がほとんど変わらない。コードサイズの変更がほとんどないということになる。

(5) 考察

アーキテクチャ情報を利用して、GCC をベースにしたクロスコンパイラを生成するコンパイラ生成系を構築できることを確認した。アーキテクチャ情報を追加変更すれば、新しい命令の追加に対応した開発システムが短時間で再構築できた。複数の評価用アプリケーションプログラムを、評価用の環境で実行速度を計測したところ、有効と思われる命令の追加が、コンパイラにも反映され、実行速度も向上できることがわかった。

ターゲット・アーキテクチャ情報の記述で不足な点はないか、などの検討が必要である。組み込み用途を考えた場合、生成コードサイズの観点から評価、考察する必要もあるだろう。その場合シンボリック命令シミュレータではなく、アセンブラと命令シミュレータが独立したシミュレータを利用する必要がある。コンパイラ生成系での、マシン固有の最適化は今後の課題である。

５．まとめ

本稿では ASIP 用 開発環境システムの開発の背景を説明し、GCC を利用したコンパイラ生成系試作について紹介した。まず、このコンパイラ生成系の概要を説明し、評価実験とその結果を述べ、次に評価実験を行い、ターゲット・アーキテクチャの変化に容易に対応できるコンパイラ生成系に適用可能であることを確認した。

今後の課題としては、試作したコンパイラ生成系の対応アーキテクチャの種類を増やし、広範囲に利用できることを評価することがあげられる。また、その際のアーキテクチャ情報を新規作成ために要した時間、あるいは変更に要した時間を評価し、利用容易性を評価することなどもある。さらに、最適化コンパイラ向け(たとえば、レイテンシやスループット情報を活用した命令スケジューリングの導入)の生成系の対応、コードサイズの評価もある。

アーキテクチャ情報記述の再検討を含め、コンパイラにとどまらず、アセンブラ、リンカ、命令セットシミュレータなどの開発用ソフトウェアの自動生成があげられる。さらに、プロファイラやアプリケーションプログラム解析系などの整備を行い、解析結果から追加したほうがいい命令を提案するアプリケーションプログラム解析ナビゲーションシステムなどが考えられる。

６．参加企業及び機関

本プロジェクトは、情報処理振興事業団による平成 13 年度未踏ソフトウェア創造事業の支援を受けて、(株) SRA 、大阪大学情報科学研究科情報システム工学専攻集積システム設計学講座により (契約件名「ASIP 用 開発環境システムの開発」) 実施されました。

７．参考文献

[1] Shinsuke KOBAYASHI, Kentaro MITA, Yoshinori TAKEUCHI, Masaharu IMAI, "Design Space Exploration for DSP Applications using the ASIP Development System PEAS-III," proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing 2002, Vol. 3, pp. 3168-3171, (May, 2002)

[2] 三田 健太朗, 小林 真輔, 武内 良典, 北嶋 暁, 今井正治, "PEAS-III システムのためのコンパイラ・ジェネレータの試作," DA シンポジウム 2001 論文集, Vol. 2001, No. 8, pp. 143-148, (2001 年 7 月)

[3] 小林真輔, 三田健太朗, 武内良典, 今井正治, "PEAS-III システムのコンパイラ生成系とその評価", 電子情報通信学会技術研究報告, VLD2001-145, vol. 101, No. 577, pp. 101-108, (2002 年 1 月)

[4] Richard Stallman, Using and Porting the GNU Compiler Collection (GCC), http://gcc.gnu.org/onlinedocs/gcc-2.95.3/gcc.html

[5] Richard Stallman, Using and Porting the GNU Compiler Collection (GCC), 矢吹洋一訳,http://www.sra.co.jp/wingnut/gcc/index.html

[6] [spim-6.4] http://www.cs.wisc.edu/~larus/spim.html1