A-10

ハード/ソフト協調設計のためのコンパイラ学習システムの設計と実現

PISHVA JOHN CYRUS P† 井出 純一† 山崎 勝弘†
†立命館大学大学院　理工学研究科

1　はじめに

近年の急速な半導体製造技術により、LSI は小型化、軽量化、高速化、そして省電力化が可能となったので、様々な機器で数多く用いられるようになってきた。これら組み込み機器は、いずれもハードウェアとソフトウェアによって構成されており、今後も広く普及することが確実視されている。我々が研究を進めているハード/ソフト協調学習システムでは、ハードウェアとソフトウェアの両分野をプロセッサの開発を通じて学習することを目的としている[1]-[3]。本研究ではこのシステムに対し、コンパイラ学習システムの開発を進めている。学習者はコンパイラの重要な構成である、字句解析部・構文解析部の定義ルール、およびコード生成部を設計することで、構成と役割を理解して学習者自身がプロセッサのコンパイラを設計できるようになることを目的としている。コンパイラの設計により、学習者は高級言語とアセンブリ言語の対応を知り、ソフトウェアの構成をより深く理解することが可能となる。また、一般的なコンパイラで行われるコード最適化の例をいくつか挙げて導入を検討する。

2　ハード/ソフト協調学習システム

ハード/ソフト協調学習システム(以下HSCS)とはプロセッサを通してハードウェアとソフトウェアの両方の知識を学習していくために考案されたシステムである。システムの構成はそれぞれの分野を学習するフローからなっており、ソフトウェアの学習フローではアセンブリプログラミングの学習から進め、プロセッサのデータパスを確認できる MONI シミュレータが用意されている。ハードウェアの学習フローでは、学習者が独自の命令セットを定義し、FPGA ボード上での検証まで行えるようにプロセッサ設計支援ツールが用意されている[1][2]。MONI とは、本研究室で MIPS のサブセットとして定義した教育用マイクロプロセッサである。

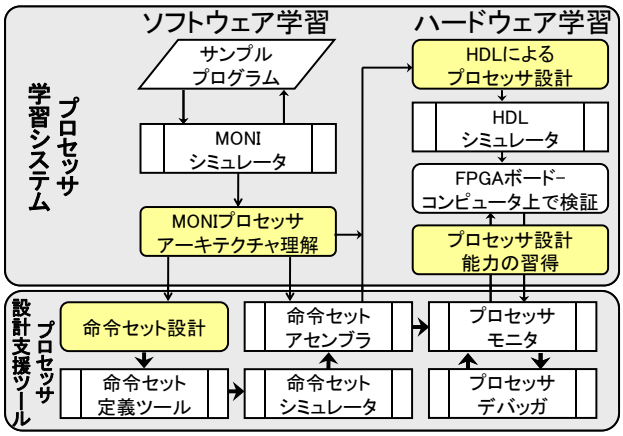

図 1　ハードソフト協調学習システム

図 1 に、ハード/ソフト協調学習システムについての学習体系を示す。ソフトウェア学習の流れは、学習者自身が用意したアセンブリプログラムを MONI シミュレータ上でシミュレーションを行い、データパスを確認しながら実行することで、プロセッサの仕組みを学習することが可能である[1]。ハードウェア学習の流れは、実際に HDL を用いて MONI プロセッサの設計、またはオリジナルプロセッサの設計を行う。次に、設計したプロセッサを HDL シミュレータによりシミュレーション検証を行い、それから FPGA ボード上に実装する。FPGA ボード上で検証する際、設計したプロセッサをプロセッサデバッガと接続し、プロセッサモニタを用いてデータを送受信することで検証を行う[1]。

3　コンパイラ学習システムの設計

3.1 システムの構成

これまで HSCS では、プロセッサを重点とした学習内容であった。ハードウェアとソフトウェアのトレードオフをより詳細に理解してもらうためには、より一般的なソフトウェアの観点からも学習できるようなコンパイラ学習システムが必要であり、MONI コンパイラの設計を行った[3]。図 2 にコンパイラ学習システムの構成を示す。学習者は字句解析、構文解析、変数表、コード生成について学習する。字句解析部と構文解析部の設計には、それぞれの定義ルールを C 言語の仕様にあわせて記述し、字句解析生成ツール Flex と構文解析生成ツール Bison を用いて字句解析部・構文解析部を生成する。変数表は、既存の変数表の処理流れを理解し、ハッシュ表をターゲットプロセッサのレジスタ数にあわせる修正を加えて利用する。最後に、生成された字句解析部・構文解析部と変数表を用いて、入力となる C ソースコードからターゲットプロセッサ上で実行可能なアセンブリコードを生成するコード生成部の設計と作成を学習者が行う。

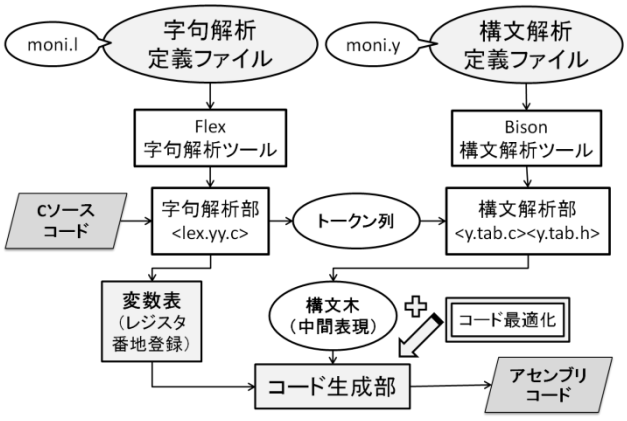

図 2　コンパイラ学習システムの構成

3.2 字句解析部

字句解析部は、ソースコードの文字列の並びをトークンの並びに変換する。トークンとは意味を持つコードの最小単位であり、キーワード、識別子、シンボル名がトークン

である。本研究ではコンパイラ・コンパイラである lex の GNU 版の Flex を用いて自動生成している。

3.3 構文解析部

構文解析部は、字句解析されたトークン列を受け取り、解析することでプログラムの構造を明らかにし、構文木を生成する。通常のコンパイラでは、この構文木だけでは解析不十分である変数型の解析を、意味解析によって構文木と意味規則の対応をとって中間表現を得る。本研究では、コンパイラの目的言語を MONI としており、MONI は変数型として整数しか扱えないので、細かい意味解析を省略し、構文解析結果をそのまま中間表現として扱う。また、字句解析と同様コンパイラ・コンパイラである yacc の GNU 版である Bison を用いて自動生成している。

3.4 変数表

変数表は、C ソースで定義された変数を配列に格納することで、その要素番号を MONI のレジスタ番号と関連づける。変数表にはハッシュ法を使用している。

3.5 コード生成部

コード生成部は中間言語から目的言語を生成する。本研究では構文解析部から生成された解析木と字句解析時に登録した変数名をレジスタ番号とし、MONI アセンブリコードを生成する。

4 コンパイラ学習システムの実現

4.1 Flex を用いた字句解析部の生成

字句解析定義ファイルとして moni.l を定義し、字句解析ツール Flex に入力することで、字句解析を行う C 言語ソースコード lex.yy.c が生成される。図 3 に示すように、字句解析定義ファイルの定義ルールは%%で区切られた 3 つのセクションで記述される。第 1 セクションと第 3 セクションには C コードを記述することで、構文解析プログラムの上部および下部に含ませることができる。

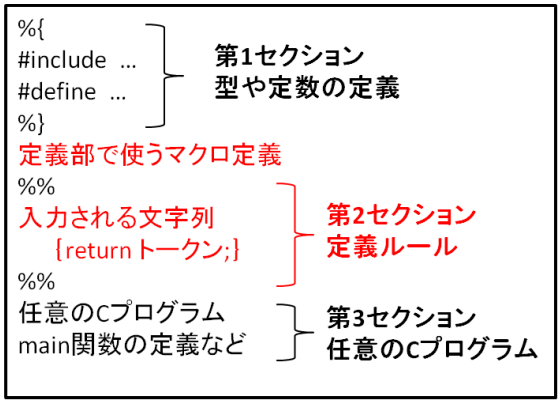

図 3 字句解析定義ファイル

第 2 セクションの定義ルールで、入力された文字列をトークンに変換するルールを書く。入力された文字列と正規表現のマッチングを取り、一致した文字列をどのようなトークンに変換するかを記述する。

字句解析定義ルールの例として、いくつかの字句のトークン定義を図 4 に示す。入力された文字列が、'+'、'-'、'*'、'/'、'='、'>'の場合、それぞれ'PLUS'、'MINUS'、'MUL'、'DIV'、'EQ'、'LC'というトークンを渡すことを示している。'if'が入力されると、yylval.h に整数を格納し、トークン'IF'を渡す。'while'も'IF'文同様にトークン'WHILE'を渡す。yylval.n に格納した値は、分岐先のラベルとして後で使用される。'{variable}'は、入力される変数を第 1 セクションで、'[a-zA-Z]'としてマクロ定義している。variable が入力として入ってくると、ポインタ型の yylval.s に変数を代入し、変数表の関数 IDentry にその代入した値を返す。そして構文解析にトークン'VARIABLE'を渡す。'[a-zA-Z][a-zA-Z0-9]*'は、英数字を表す。英数字が入力されれば、字句解析部は英数字を yylval.s に代入し、yytext に英数字をコピーして、トークン'NAME'を構文解析に渡す。最後の'[0-9]+'は整数を表す。同様に整数を yylval.s に代入し、yytext に整数をコピーして、トークン'NUMBER'を構文解析に渡す。

```
%{...
%}
mul             [*]
div             [/]
%%
+               {return(PLUS);}
-               {return(MINUS);}
{mul}           {return(MUL);}
{div}           {return(DIV);}
=               {return(EQ);}
>               {return(LC);}
If              {yylval.n=++n;return(IF);}
while           {yylval.n=++n;return(WHILE);}
{variable}      {yylval.s = IDentry(yytext, yyleng);
                return(VARIABLE); }
[a-zA-Z][a-zA-Z0-9]*  {yylval.s=strdup(yytext);
                        return(NAME);}
[0-9]+          {yylval.s=strdup(yytext);
                return(NUMBER);}
・・・
%%
```

図 4 字句のトークン定義

トークン列の生成例を図 5 に示す。入力された文字列が順に、'N'、'M'が[NAME]、'='が[EQ]、'+'が[PLUS]、'0'、'1'が[NUMBER]としてトークンに変換されて出力されている。'if'や'while'以降も同様に字句解析部で定義したルールに従いトークンを生成している。

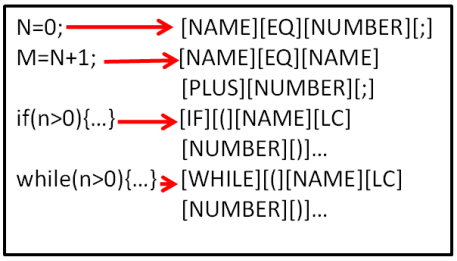

図 5 字句の解析結果

4.2 Bison を用いた構文解析部の生成

Flex を用いた字句解析プログラム生成の流れと同様に、構文解析定義情報の moni.y を Bison に与えることで、構文解析プログラムの y.tab.c と y.tab.h を生成する。y.tab.h には、ユーザーが割り当てたトークン番号と宣言したトークン名に対応させる#define 文を含んでいる。

構文解析定義ルールは第 2 セクションの定義ルール以外は、字句解析定義ルールと同様の構成である。 定義ルールは、入力されたトークンに対し、それぞれ記述された処理を行う。Bison は、BNF（バッカス・ナウア記法）に似た構文規則に基づいてパーサを生成する。

構文解析定義ルールの例として、if 文と while 文と式の構文定義を図 6 に示す。

```
%%
if:             '(' expr ')' SEP {printf("¥t%02dF",$1);}
                '{' expr ';' expr ';' '}'
                {printf("¥t%02dF",$1);}
                ;
while :         WHILE {printf(" %02dT:¥n",$1);}
                '(' expr ')'
                {printf("¥tSEP %02dF¥n",$1);}
                '{' expr ';' expr ';' '}'
                {printf("¥tJUMP %02dT¥n %02dF:¥n",$1,$1);}
                ;
expr : VARIABLE  {printf("¥t$%s",$1);}
       | expr PLUS expr {print("¥tADD");}
       | expr MINUS expr {print("¥tSUB");}
       | expr EQ expr {print("¥tEQ");}
       | expr LC expr {print("¥tLC");}
...
%%
```

図 6　if 文、while 文、式の構文定義

expr は、変数や整数における式、論理演算子の定義である。まず'if'の処理について説明する。上図の'if'における記述はソースコードで書かれる'if'文「if（条件）（式）」の形にマッチする。トークン‘IF’を受け取れば、（条件）に値する expr が入力され、その（条件）に対応する中間表現の命令と、（条件）が偽の場合に分岐するラベルを設置する。次に（式）に対する expr が入力され、expr の定義による処理が施される。最後に（条件）が偽の場合のラベルを生成する。次に'while'の処理について説明する。if 文と同様に、'while'文「while（条件）（式）」の形にマッチする。トークン‘WHILE’を受け取り、printf 関数によってループ先のラベルを出力する。次に（条件）と（式）に対する処理が施される。'while'文の（式）に対する処理が終われば、ループするために無条件分岐の JUMP を生成する。expr については、expr の中が字句解析部で定義した変数、すなわち 'VARIABLE' であれば、字句解析部で yytext に登録した変数の値を出力する。

構文解析結果の例を図 7 に示す。

```
N=0;                    $N #0 EQ
M=N+1;                  $M $N #1 ADD EQ
If (n>1){...}           $n #1 LC SEP 01F
                        ...
                01F:
while(n>0){...} 02T:    $n #0 LC SEP 02F
                        ...
                        JUMP 02T
                02F:
```

図 7　構文の解析結果

それぞれの出力は後置記法に似た構文情報を出力している。'if' 文は処理の外（01F）にラベルを設置し、括弧内の条件式が一致しなかった場合に処理の外にジャンプして処理を行わないことを表す。'while'文は'while'文の先頭（02T）、および'while'文の外（02F）にラベルを出力し、括弧内の条件式が一致する場合に処理を実行してループし、一致しなくなった場合に'while'文の外のラベルに分岐してループを抜ける処理を表している。

4.3 変数表の登録

変数表にはハッシュ法を用いた。ハッシュ法にはいくつか技法があるが、本研究における変数表はチェイン法を用いた。ハッシュ表を H、与えられた文字列に対するハッシュ値を k とする時、その文字列を持つポインタを H[k]に入れる。与えられた異なる文字列が、同じハッシュ値に入ることを衝突という。チェイン法では、そのまま同じ位置に格納しようとするが、代わりに連結リストを用意し、データを連結する。このリストが衝突チェインである。変数表の構成を図 8 に示す。

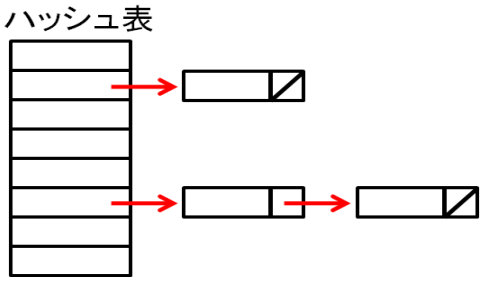

図 8　変数表の構成

データを保存する領域はハッシュ表とは別の領域に確保されている。本研究では、MONI アーキテクチャの汎用レジスタの数(8 つ)とハッシュ表を合わせている。 このハッシュ表を用いて C ソースコードの変数を変数表に登録し、その要素番号をコード生成部に渡すことでレジスタ番号を決定する。

4.4 コード生成部の設計

コード生成部の役割は、構文解析から生成された中間表現と、変数表に登録された要素番号を用いて MONI アセンブリコードの生成を行う役割を果たす。中間表現から MONI アセンブリコードの生成例を図 9 に示す。

構文解析結果を本研究では中間表現として扱う。コード生成部で一度 MONI アセンブリコードの形に直し、変数表に登録した変数の値とレジスタ番号の変換を行うことで MONI アセンブリコードを生成している。中間表現を

図の 2 つ目の形式にしなかったのは、学習者がコード生成部を設計する際、学習者自身の命令セットが 2 オペランドの場合も考慮し、学習者にとってより自由に設計できるように本研究では構文情報をそのままに置いている。

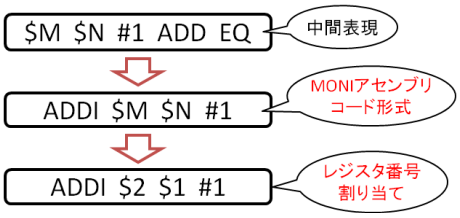

図 9 <M=N+1>のコード生成例

5 コード最適化の検討

5.1 ループ最適化

ループ処理はその部分のプログラムを何度も繰り返し処理を行う。特に画像処理では、ループ処理が二重にも三重にもネストされて計算量が多くなる。そのためループ処理の最適化を行うことで、計算量を大きく削減することが期待できる。

ここでは「共通式の削除」と「不変式の移動」の 2 つの最適化を検討する。共通式の削除は、同一の計算を複数回行っているところを見つけて、その計算を 1 度だけ行い、変数に代入しておくことで、2 回目以降は計算された変数を参照する。不変式の移動は、ループ内で変化しない計算を見つけ、そのコードをループの外に移動することでループ内の不要な計算量を減らすことができる。これら 2 つの最適化はループ処理内の配列のアドレス計算処理の削減において効果的である。

5.2 命令スケジューリング

プロセッサによっては 1 クロックサイクルで複数の命令実行を行うことがある。このとき、分岐命令や依存関係が検出されると、ストールが発生し、一部の命令の実行が後回しにされてプロセッサの性能が発揮できなくなってしまう。

ここでは「ループ展開(ループアンローリング)」の最適化を検討する。ループ展開とは、ループ内の処理を展開し、ループの処理回数を減らす最適化である。ループ内の処理が単純なコードの場合、ループ制御の分岐命令が妨げになる。そのため、処理を展開してループ制御の処理を減らす、または処理をすべて展開してループ処理そのものをなくすことで、プロセッサの性能が発揮できるようになる。

ループ展開の簡単な例を挙げる。図 10 の(1)に示すコードにループ展開を行ったコードを同図の(2)に示す。展開後のコードでは、ループ内の処理を 2 つにし、ループの回数を半分に減らすコードを示している。それぞれのコードの下にはパイプライン上で実行した場合の処理の一部を示している。

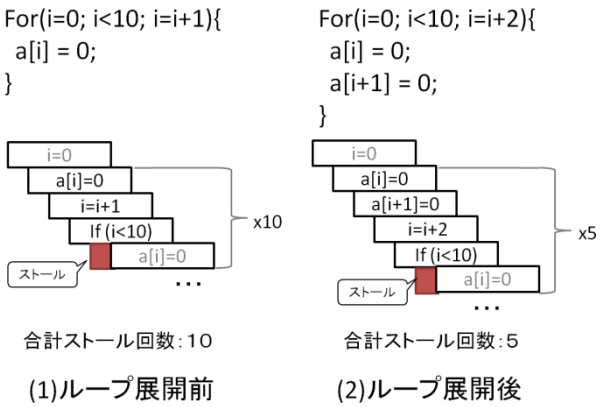

図 10 ループ展開前後のコードとそのパイプライン処理

1 命令の実行を 5 クロック、条件分岐で発生するストールを 1 クロックと仮定すると、'if (i<10)' の条件コードを実行するたびに 1 度ストールが発生する。このループ処理の全体を処理すると、展開前に 10 回発生するストールが展開後では半分の 5 回に減少する。このコードの場合にループ展開を行うことで、ループ全体の処理数が 4 割程度減少する。

ループ展開はパイプラインに限らず、スーパースカラーや現在主流のマルチコアにも効果を発揮する。たとえば 4 つのコアを持つマルチプロセッサなら、依存関係のない 4 つの命令を並列に処理するように展開することでプロセッサの並列性能を発揮できる。

6 おわりに

本研究では、ハード/ソフト協調学習システムにおけるコンパイラ学習システムの設計と実現について述べた。また、コンパイラによるコード最適化について検討した。今後、コンパイラ学習システムにコード最適化部を導入し、学習者がコンパイラの役割をより詳しく学習できる環境の実現を目指す。

参考文献

[1] 難波翔一郎，志水建太，山崎勝弘，小柳滋：プロセッサ設計支援ツールの設計・実装とハード/ソフト協調学習システムの評価，FIT2007，情報科学技術レターズ，LC-002，2007.

[2] 志水建太，井手純一，山崎勝弘：プロセッサ設計教育における命令セット定義ツールと命令セットシミュレータの試作，情報処理学会，関西支部大会講演論文集，A-05，2008.

[3] 井手純一，志水建太，山崎勝弘：ハード/ソフト協調学習のためのコンパイラ開発の検討，情報処理学会，第 71 回全国大会論文集，2K-3，2009.

[4] 中田育男：コンパイラの構成と最適化（第 2 版），朝倉書店，2009.

[5] 原田賢一：コンパイラ構成法，共立出版，1999.