3J-02

FPGA を指向した CtoHDL コンパイラの開発

長尾 圭
高木 正昭, 丸山 勉, 星野 力
筑波大学理工学研究科

1 はじめに

近年、Field Programmable Gate Array(FPGA) の大規模化や高速化に伴い、FPGA を用いた研究が多く行われている。その中で、FPGA を既存のプロセッサと組み合わせて、アクセラレータとして用いる応用が考えられている。

FPGA は、ある特定の問題に対して、計算の並列化やパイプライン化を行う事により高速化を計ることが出来る。その設計作業は、主にハードウェア記述言語 (HDL) を用いた設計が主流であるが、HDL のプログラミングにより、問題を設計することは、高級言語である C 言語に比べて容易ではない。

そこで、プログラミング作業の軽減のために、C 言語のソースを最適な HDL コードに変換するコンパイラの開発を行う。

2 C to HDL コンパイラ

2.1 コンパイラの構成

コンパイラの構成は、字句解析、構文解析、中間コード生成、最適化から成る。中間コードは、C 言語における各変数の依存関係を、計算順序の依存関係をもたない HDL に変換するさいに、一次的に、各依存関係を記述するために使用される。さらに、中間コードは、高速化のための最適化（並列化、パイプライン化）を施すために使われる。

A C to HDL compiler for FPGA
Kei Nagao(University of Tsukuba)
Masaaki Takagi,Tsutomu Maruyama,Tsutomu Hoshino

```
1: int i=0,j=0,k=0,hojin[(N+1)*(N+1)];
2: while(k<=N*N){
3:         if(i==N){
4:           i=1;
5:           j=j+1;
6:         }else if(i>0){
7:           i=i+1;
8:         }else{
9:           i=1;
10:          j=1;
11:        }if(j%4==i%4 || (j%4+i%4)%4==1
12:          hojin[k]=(N+1-i)*N-j+1;
13:        else
14:          hojin[k]=(i-1)*N+j;
15:        k=k+1;
16:        }
```

図 1: 4N 魔法陣

コード	フロー依存	出力依存	条件
(1) i=0			
(2) j=0			
(3) k=0			
(4) k<=N*N			
(5) !=N			(4)
(6) i=1		(1)	(4)&(5)
(7) !=1 && !=6			(4)&(5)
(8) j=1			(4)&(5)
(9) j==8		(2)	(4)&(5)
(10) j==2||=9			(4)&(5)
(11) i=0			(4)&(5)&
(12) i=1			(4)&(5)&(!(1))
(13) i=i+(2)		(7)	(4)&(5)&(!(1))
(14) i==5||=7			(4)&(5)&(!(1))
(15) i=-1		(14)	(4)&(5)&(3)(11)
(16) !=3 && !=4			(4)&(5)&(3)(11)
(17) i=1		(16)	(4)&(5)&(3)(11)
(18) j==5 && !=10			(4)&(5)&(3)(11)
(19) j%=4			(4)
(20) j%=4			(4)
(21) i=19+=20	(19)(20)		(4)
(22) j%=4			(4)
(23) j%=4			(4)
(24) i=22+=(23)	(22)(23)		(4)
(25) i%=4	(24)		(4)
(26) i=25+=1	(25)		(4)
(27) i=24+=26	(24)(26)		(4)
(28) N+1			(4)&(27)
(29) i=28-i	(28)		(4)&(27)
(30) i=29*N	(29)		(4)&(27)
(31) i=30+j	(30)		(4)&(27)
(32) i=31+1	(31)		(4)&(27)
(33) hjn[k]=i-32			(4)&(27)
(34) k+1			(4)&(27)
(35) i=1=N	(34)		(4)&(27)
(36) i=35=j	(35)		(4)&(27)
(37) hjn[k]=-36)		(33)	(4)&(27)
(38) hjn[k]=i-j+37)			(4)&(27)
(39) k+1			(4)
(40) k=39)		(3)	(4)
(41) k=k+(3)(39)			(4)

図 2: 4N 魔法陣の中間コード表

2.2 中間コード生成

まず中間コードを説明するために、図 1の 4N 魔法陣を例に挙げる。中間コードの生成は以下の順序で行われる。まず与えられたプログラムから、a=b+c の様なオペランド（この式でいう a,b,c）から成る計算項ごとに分割し、それを基に、図 2に中間コード表を作成する。中間コード表には、図 2のように、中間コード列の番号と、分割した計算項ごとの、その他の計算項との依存関係（フロー依存、逆依存、出力依存）が記述される。

逆依存は、代入を行うたびに名前を付け直すことで依存関係を解消できるため、検索する必要はない。フロー依存と出力依存はそれ以前の中間コード列を検索し、現在のオペランドに対して、他に代入が行われているかどうかを調べ、図2のコード番号(6)のように、他に代入が行われていたら出力依存とし、例えばコード番号(29)のように参照が行われていたらフロー依存とする。出力依存が検出された場合、条件により代入値を選択するselector（sel）を加え、それ以降の依存関係には、selectorのコード番号が用いられる。

分岐命令によって、オペランドの値が確定できない場合には、分岐命令後に計算する必要がある。このため、どの条件ブロックに属するかを、記録する必要がある。ステージ同じオペランドに対して、複数の依存関係がある場合、同じ条件ブロック内の依存関係を優先する。

2.3 最適化

2.3.1 並列化

並列化の手順を以下に示す。ステージとは、コード番号から選び出した同時に実行できる中間コードのグループである。便宜上このステージに実行順番をつけて、プログラム全体の計算過程を見るために使用している。

1. 中間コード表において、依存関係が記述されていないコード番号を選び出し、それらのグループを第1ステージとする。
2. 次に、1のグループの実行によって、依存関係が解消されるコード番号を選び出し、そのグループを第2ステージとする。
3. 以上の作業を全てのコードが選び出されるまで続ける。

並列化を行うと、図3のように6ステージで全ての計算を終わらすことができる。

2.3.2 パイプライン化

パイプライン化を行うことによって最大パイプライン段数分の高速化を実現することができる。図1のプログラムだと6段で実行可能なため、通常6*N*N回の計算が必要な所、パイプラインで実行すると、N*N+5回の計算で実行することが出来る。図1のプログラム中には含まれていないが、フィードバック変数を含んだプログラムや、1ループ中に2回以上同一のメモリへのアクセスがあるプログラムの場合、同一変数中1番最後に計算やメモリアクセスが行われた後、次のループでのアクセスを行うことにより、パイプライン化を実現している。

ステージ	コード番号
1.	(1)(2)(3)(4)(5)(6)(8)(11)(12) (15)(17)(19)(20)(22)(23)(28) (34)(39)　(9)(10)(14)(16)(18)
2.	(21)(24)(29)(35)
3.	(25)(30)(36)
4.	(26)(31)(37)
5.	(27)(32)
6.	(33)　(38)

図 3: 並列化されたコード番号

3 おわりに

本研究では、FPGAアクセラレータのための、C to HDLコンパイラについて述べ、並列化、パイプライン化の方法を提案した。コンパイラ自体は作成中である。CtoHDL変換のアルゴリズムに基づいて、4N魔法陣を、手書きでHDLに書き下した結果、約50MHzで実行することが出来た。パイプラインも完全に動き、6段のパイプラインを構成することが出来た。

4 参考文献

[1] 高木正昭、丸山勉、星野力(1999): FPGAを用いた木探索問題の高速化, 情報処理学会第58回全国大会講演論文集, 129-130 [2] 五月女健治: yacc/lexプログラムジェネレータ on UNIX [3] 若松一敏(1996/2/12): 日経エレクトロニクス, 伝送用LSIを動作合成で開発, 機能設計の期間が1/10に短縮 [4] 末松敏則(1999/8): 特集やわらかいハードウェア, 情報処理学会誌