Vol.2018-ARC-230 No.38
Vol.2018-SLDM-183 No.38
Vol.2018-EMB-47 No.38
2018/3/8

静的ソース解析を用いたデータ転送量見積もりに基づく FPGA アクセラレータ設計支援

一場 利幸[1,a)]　田宮 豊[1]　渡部 康弘[1]　上原 義文[1]

概要：高性能な FPGA アクセラレータを実現するためには，処理の実行時間だけでなくデータ転送時間を見積もった上で，アーキテクチャ設計を行う必要がある．そのため，データ転送量の見積もりが重要であるが，もし無駄なデータを含んでいると，正しいデータ転送時間を見積もれず，適切なアーキテクチャ設計が行えない．そのため，正味のデータ転送量を見積もることが必要である．正味のデータ転送量の見積もりは，人間がソースコードを読んで解析した結果を利用しており，工数がかかる問題がある．本稿では，正味のデータ転送量の見積もりを目的とした，ソースコード解析について述べる．正味のデータ転送量を見積もるために必要な情報を挙げ，コンパイラ基盤 LLVM を利用して解析可能であることを示す．そして，得られた情報からデータ転送量を見積もる方法を提案し，C 言語の画像フィルタプログラムを例に，正味のデータ転送量見積もりが行えることを示す．

1. はじめに

ムーアの法則に従った半導体の微細化が限界を迎え，微細化による CPU の性能向上が緩やかになっている．そのため，CPU で実行されるソフトウェア処理の一部を，FPGA にオフロードしてシステム全体の性能を向上する方法が注目されている．FPGA をアクセラレータとして使う代表的なシステム構成は，図 1 のような構成である．

FPGA アクセラレータを活用して高速化を実現するには，FPGA にオフロードする処理の選択や，その処理モジュールの並列化やパイプライン構成の構造設計など，全体アーキテクチャの設計が重要である．アーキテクチャの設計においては，各処理の実行時間だけでなく，処理間のデータ転送時間も考慮する必要がある．これは一般に，処理の実行時間に比べて，データ転送時間が無視できないためである．例えば，FPGA の処理時間に比べて，CPU と FPGA 間（図 1 の PCIe）のデータ転送の時間が長い場合，FPGA にオフロードしても高速化の効果は低い．このように，FPGA アクセラレータを活用した高速化を検討する場合，データ転送時間を求めるために処理間のデータ転送量を見積もることは重要である．

ここで，適切なアーキテクチャ設計のために必要となるのは，変数や配列などへの単純なアクセス量ではなく，同じデータの 2 度読みなどの重複アクセスを除いた正味のデータ転送量である．さらに，アーキテクチャ設計ではハードウェア化した際のモジュールやローカルメモリの構成を検討する必要があるが，一般にソフトウェアはこれらを意識した記述になっていないため，処理間（モジュール間）の正味のデータ転送量を見積もることは難しい．

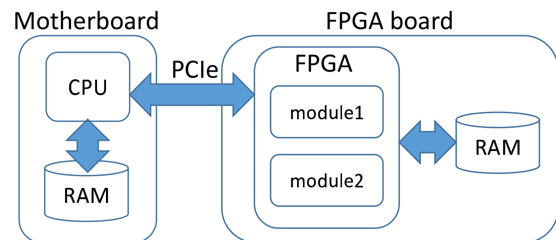

図 1　システム構成

通常，データ転送量を見積もる作業は，人間がソースコードを読んで解析した結果と，ソフトウェアを実行したプロファイル結果とを組み合わせて行われている．組み合わせる理由は，ソースコードだけでは入力データに依存して決まる情報は解析が困難であり，一方，プロファイル結果だけでも網羅的な解析が困難であり，一長一短なためである．本研究では，人間がソースコードを読む工数を削減する技術として，ソースコードの静的解析に着目した．既存のコンパイラ技術によってソースコードの様々な情報が解析可能であるが，正味のデータ転送量の見積もりを目的としたツールは著者の知る限り存在しない．なお，プロファイル結果を用いて，FPGA アクセラレータの設計を支援する情

[1]　（株）富士通研究所
Fujitsu Laboratories, Ltd.
[a)]　t.ichiba@jp.fujitsu.com

© 2018 Information Processing Society of Japan

報を可視化する取組みとして [1] が存在する．

データ転送量として大きな割合を占めるのは，配列や構造体などの複合型データであるため，本研究では配列に着目する．配列のデータ転送量について，ソースコードからコンパイラ基盤 LLVM[2] を用いて解析する方法を述べる．そして，C 言語の画像フィルタプログラムを例に，正味のデータ転送量の見積もりが行えることを示す．

本稿の構成は次の通りである．まず，2 章でアーキテクチャ設計における正味のデータ転送量の意義と静的解析について述べ，3 章でデータ転送量の見積もりを実現する方法を説明する．4 章で見積もり方法の評価を行い，5 章でまとめと今後の課題を述べる．

2. データ転送量と静的コード解析

2.1 正味のデータ転送量

この節では，まず，アーキテクチャの設計において，正味のデータ転送量を見積もる意義について述べる．そして，正味のデータ転送量について，具体例を用いて述べる．

アーキテクチャの設計では，各処理の実行時間とデータ転送時間の見積もりに基づいて，並列化やパイプライン構成の構造設計などを行う．そのため，ハードウェア化した際のモジュールに相当する処理の分割や，モジュール間のデータ転送時間の見積もりが重要となる．データ転送量からデータ転送時間を見積もるが，もし無駄なデータが含まれていると，正しいデータ転送時間を見積もれず，適切なアーキテクチャ設計が行えない．そのため，無駄なデータを含まない正味のデータ転送量を見積もることが重要となる．ここでは，処理の分割について，モジュールに相当する処理について述べたが，CPU で実行する処理と FPGA へオフロードする処理の分割についても，同様に，正味のデータ転送量を積もることが重要である．

ここで，正味のデータ転送量について述べる前に，データ転送量について述べる．データ転送量は，ある処理から別の処理に対して転送するデータ量である．処理内でアクセスが閉じているデータは，他の処理に転送する必要がないため，データ転送量としてカウントしない．例えば，ある関数を 1 つのモジュールに対応させたとき，関数内のローカル変数はデータ転送量としてカウントしない．ここで，その関数を 2 つのモジュールに分けて，あるローカル変数を両方のモジュールでアクセスしているときは，そのローカル変数を転送する必要があるので，データ転送量としてカウントする．このように，データ転送量は処理の分割方法によって変化する．

そして，正味のデータ転送量とは，同じデータへの複数回アクセスや処理内でアクセスしないデータを除いた，データ転送量である．同じデータへの複数回アクセスは，そのデータをバッファで保持しておけば再度転送する必要がない．また，実際にアクセスしないデータを転送するのは明らかに無駄である．これらの無駄なデータの転送を除いたものが正味のデータ転送量である．

```
 1 # define S 10
 2 # define X 10
 3 void foo(int in[S][S], int out[S]) {
 4   int i, j;
 5   loop_i: for(i=0; i<X; i++) {
 6     out[i] = 0;
 7     loop_j: for(j=0; j<X; j++) {
 8       out[i] += in[i][j]*in[j][i];
 9     }
10   }
11 }
```

図 2 データ転送量を説明するコード例

本研究では解析していないが，データのビット長にも無駄のある可能性がある．例えば 0 か 1 かの 1 ビットしか利用しないデータを，ソフトウェアで 32 ビットの int 型を使って記述されていた場合に，31 ビットの転送は無駄である．最適なビット長を使うことで，ハードウェアの規模を減らすことができる．ビット長の解析は既存研究 [3] が存在しており，これらを取り入れることで，正味のデータ転送量を求めることができる．

データ転送量の具体例を，図 2 のコードを用いて述べる．内側のループ（loop_j）に対しては，配列 in は [i][0] から [i][9] までの 10 個と [0][i] から [9][i] までの 10 個を read する．合わせると内側のループ 1 回あたり read を 20 回行い，それが外側のループ（loop_i）で 10 回繰り返されるため，トータルでは read を 200 回要する．一方，外側のループに対しては，配列 in は 1 度転送すれば良い．配列 in のインデックスは各次元で 0 から 9 まで変化するので，100 個の要素を転送する．そのため，外側のループについて考えると，read は 100 回行えばよい．内側のループでは，配列 in の要素を複数回転送する必要があるが，外側のループでは，1 度転送すればよい．これが意味するところは，ハードウェア化した際のメモリをどこに持たせるかによって，データ転送量が変わるということである．例えば，内側のループ内の演算処理だけをオフロードすると，200 回の read に相当するデータ転送が必要であるが，処理で必要な正味のデータとしては 100 回の read に相当するデータ転送で済むということである．ただし，この場合は，データを保持するためのバッファがモジュール内に必要である．

2.2 見積もりに必要な情報

配列のデータ転送量を見積もるために必要な情報は，以下の通りである．

(1) 配列へのアクセスの種類（read か write か）

(2) ループ変数の式で表現された，配列のインデックス

(3) ループ変数の取り得る値

```
1  {"statements" :[
2    {"accesses" :[
3        {"kind" :"read",
4          "relation" :"{ Stmt_for_body3[i0, i1] ->
               MemRef_in[i0, i1] }"
5        },
6        {
7          "kind" :"read",
8          "relation" :"{ Stmt_for_body3[i0, i1] ->
               MemRef_in[i1, i0] }"
9        },...
10     ],
11     "domain" :"{ Stmt_for_body3[i0, i1] :i0 <= 9
            and i0 >= 0 and i1 <= 9 and i1 >= 0 }",
12     "name" :"Stmt_for_body3",
13     "schedule" :"{ Stmt_for_body3[i0, i1] -> [i0
            , 1, i1] }"
14   },...
15 ]}
```

図 3　Polly の JSON 出力例（配列 in について抜粋）

(4) ループ回数

まず，配列へのアクセスが read か write か，もしくは両方なのかを判別する必要がある．そして，配列の要素ごとにアクセスするかどうかを調べる必要がある．つまり，配列アクセス時のインデックスを調べる必要がある．配列アクセスは，ループ処理内で行われ，インデックスはループ変数によって決まることが多い．そのため，配列のインデックスをループ変数の式で表現し，ループ変数の取り得る値がわかれば，どの要素にアクセスするかがわかる．また，図 2 のようにループがネストしていたときは，トータルのデータ転送量を求めるためにループ回数の解析も必要である．

これらの情報があれば，前述したように，配列のデータ転送量を求めることができる．

2.3　LLVM/Polly によるコード解析

2.2 節で述べた見積もりに必要な 4 つの情報は，一般的なソフトウェアのコンパイラ最適化においても重要な情報である．これらの情報は，解析可能なソースコードに制約があるものの，多面体モデルを用いた解析処理によって取得できる．LLVM の場合は，多面体モデルを用いた解析・最適化処理は，Polly[4] というツールで実装されている．本研究では，Polly から見積もりに必要な情報を得ることとした．解析できるソースコードの制約については，文献 [5] に記述されている．

例えば，図 2 のコードを Polly[*1] を使って解析すると，JSON 形式で図 3 のような結果を得られる．図 3 から得られる情報は，表 1 のとおりである．Polly の解析結果では，外側ループのループ変数を i0，内側ループのループ変数を i1 としており，ネストが増えると数字が増える．図 2 の例では i と i0，j と i1 がそれぞれ一致しているが，一般的には i0 と i1 は 0 から始まり，1 ずつ増えるよう正規化される．なお，Polly の解析結果で表現されている情報の意味については，文献 [6] により詳しい情報が記述されている．

*1　LLVM/Polly のバージョンは 3.8.1 を利用

表 1　図 2 の Polly 解析結果

解析対象ループ	ループ変数と範囲	アクセス	配列名	インデックス
loop_i	$0 \le i0 \le 9$	out	out	[i0]
loop_j	$0 \le i0 \le 9$, $0 \le i1 \le 9$	read	in	[i0][i1]
		read	in	[i1][i0]
		read	out	[i0]
		write	out	[i0]

3.　配列のデータ転送量見積もり方法

ここでは，Polly の解析結果を用いて，配列のデータ転送量を見積もる方法について述べる．ループ変数の取り得る値から，配列のインデックスを解析する際に，具体的な値を列挙する方法と，変数で表現して代数的に取り扱う方法がある．この 2 つは，正味のデータ転送量を見積もれる条件や処理時間が異なるため，用途に応じて使い分けることが望ましい．2 つの方法について述べた後，正味のデータ転送量を見積もれるかどうかで，方法を切り替えるハイブリッド法についても述べる．

3.1　見積もり方法 1

見積もりに必要な情報である図 3，表 1 を得てから，データ転送量を見積もる方法を述べる．

表 1 より，配列 in については，2 つの read（in[i0][i1] と in[i1][i0]）があり，i0 と i1 の取り得る値が $0 \le i0 \le 9$, $0 \le i1 \le 9$ という範囲であることが分かる．配列 in について，トータルのデータ転送量を見積もるには，各次元のインデックスについて，この範囲内で取り得る値を数えれば良い．これは，ループ変数の式で表現された配列のインデックスに対して，ループ変数の取り得る値を制約として与えた，制約充足問題（Constraint satisfaction problem, CSP）であるため，CSP ソルバを用いて解くことができる．

また，内側のループに対してデータ転送量を見積もる場合は，対応するループ変数 i1 を定数（例えば 0）とみなして，同様に CSP を解けばよい．

この見積もり方法は，実装が容易であるが，ループ変数の取り得る値が具体的な定数として分かっている必要がある．

3.2　見積もり方法 2

ループ変数の取り得る値が静的にわからない場合，例えば図 2 で X がマクロではなく，foo 関数の引数となっていた場合（ただし，$X <= S$ とする）は，前述の見積もり方法 1 ではデータ転送量を求めることができない．以降，X

© 2018 Information Processing Society of Japan

が foo 関数の引数となっていた場合を用いてデータ転送量を見積もる方法を述べる．

ループ変数は，0 から始まり 1 ずつ増えるよう正規化されるため，上限が変数を含む式となっているケースを考える必要がある．例では，X である．つまり，配列 in については 2 つの read（in[i0][i1] と in[i1][i0]）があり，i0 と i1 の取り得る値は $0 \leq i0 \leq X-1$，$0 \leq i1 \leq X-1$ という範囲である．in[i0][i1] のインデックス i0 について見ると，下限は 0 で上限は X-1 であるから，i0 の取り得る値は X パターンある．i1 も同様である．したがって，in[i0][i1] は X^2 の read のデータ転送量と見積もれる．in[i1][i0] も同様であり，トータルで配列 in については $2X^2$ の read のデータ転送量と見積もれる．

この見積もり方法は，変数を取り扱うことができるが，インデックスの下限と上限からデータ転送量を見積もるため，正味のデータ転送量を見積もるには制約がある．例えば，図 2 では，ループ変数 i にたいして in[2*i] というアクセスとなっている場合は，配列 in の全てのデータにアクセスせず，下限と上限から正味のデータ転送量を見積もることはできない．

3.3 ハイブリッド法

見積もり方法 2 で正味のデータ転送量を求められるのは，配列のインデックスの下限から上限までの範囲で，全ての要素をアクセスする場合のみである．そこで，見積もり方法 2 で正味のデータ転送量を求められるかどうかで，見積もり方法 1 と 2 を切り替えるハイブリッド法について検討した．

見積もり方法 2 で正味のデータ転送量を見積もれないケースを説明する．Polly で解析可能なソースコードの制約から，配列のインデックスは，定数とループ変数の線形結合で表現可能 [5] であるので，$\sum_j k_j * i_j + n$ のように表現できる（j はループ階層の深さ，i_j はループ変数，k_j, n は定数）．また，ループ変数は正規化されるので，1 ずつ増える．そのため，ループ変数 i_j が 1 つ増えるとインデックスは k_j だけ増える．このことから，ループ j については，k_j の絶対値が 1 より大きい場合にインデックスが連続しない．

ハイブリッド法は，このような場合に見積もり方法 1 に切り替えることにより，正味のデータ量を適切に見積もれるようにする．

4. 実装と評価

見積もり方法 1 と 2 を実現する Python スクリプトを開発した．これは，Polly の解析結果を入力として，各配列のデータ転送量をループの階層ごとに求める．Polly の解析結果は，JSON 形式による出力が可能であるため，この JSON ファイルを読み込む．見積もり方法 1 では，CSP を解くために Google Optimization Tools[7] を利用している．なお，開発した Python スクリプトは 1 スレッドで動作する．

```
1  #define S 100
2
3  int kernel[3*3] = /* 横方向のSobelフィルタ */
4   { −1, 0, 1, −2, 0, 2, −1, 0, 1 };
5
6  void filter(int in[][S], int X, int Y, int out[][S]) {
7    int i, j;
8    int ii, jj;
9    int iii, jjj;
10   int temp;
11
12  loop_i: for(i=1; i<Y−1; i++) {
13   loop_j: for(j=1; j<X−1; j++) {
14          temp = 0;
15     loop_ii: for(ii=0; ii<3; ii++) {
16          loop_jj: for(jj=0; jj<3; jj++) {
17              iii = i−1+ii;
18              jjj = j−1+jj;
19              temp += kernel[ii*3+jj] * in[iii][jjj];
20            }
21          }
22          out [i][j] = temp;
23     }
24   }
25 }
```

図 4 画像フィルタプログラム

図 4 の画像フィルタプログラムに対して，Polly で解析を行うと表 2 の結果が得られる．Polly の解析結果を JSON ファイルで読込み，Python スクリプトでデータ転送量の見積もりを行った．

まず，見積もり方法 1 と 2 について，各配列のデータ転送量をループの階層ごとに求めるのにかかる時間を計測した．OS は CentOS 7.4，CPU は Intel Core i7-6700K，メモリ容量は 64GB の実行環境で計測を行った．見積もり方法 1 は，ループ変数の取り得る値について，全ての組合せを列挙して，配列のインデックスを求める実装となっている．そのため，X と Y の値が大きくなると，列挙する組合せが増え，実行時間が遅くなってしまう．見積もり方法 2 は，X と Y を変数として扱うので一定時間で実行が終わる．具体的には，X と Y の値が 100 のときは，見積もり方法 1 では 38 秒程度かかるが，見積もり方法 2 では 2 秒以下である．図 4 のコードでは，見積もり方法 2 で正味のデータ転送量が求められるため，ハイブリッド法でも見積もり方法 2 により一定時間で実行が終わる．

Polly の解析結果から，データ転送量を見積もった結果は表 3，表 4 のとおりである．表 3 は，最も外側のループ loop_i についてデータ転送量を見積もった結果である．見

表 2 図 4 の Polly 解析結果

ループ変数と範囲	アクセス	配列名	インデックス
$0 \leq i0 \leq Y-3$, $0 \leq i1 \leq X-3$, $0 \leq i2 \leq 2$, $0 \leq i3 \leq 2$	read	kernel	[3*i2+i3]
	read	in	[i0+i2][i1+i3]
	write	out	[1+i0][1+i1]

表 3 データ転送量の見積もり結果（loop_i についてのみ）

見積もり方法	配列名	データ転送量
1	kernel	9
	in	10000
	out	9604
2	kernel	9
	in	XY
	out	(X-2)(Y-2)

表 4 見積もり方法 2 による見積もり結果

解析対象ループ	解析対象ループ変数	配列名	データ転送量	
			解析対象内	filter 関数全体
loop_i	i0, i1, i2, i3	kernel	9	9
		in	XY	XY
		out	(X-2)(Y-2)	(X-2)(Y-2)
loop_j	i1, i2, i3	kernel	9	9(Y-2)
		in	3X	3X(Y-2)
		out	X-2	(X-2)(Y-2)
loop_ii	i2, i3	kernel	9	6(X-2)(Y-2)
		in	9	9(X-2)(Y-2)
loop_jj	i3	kernel	3	6(X-2)(Y-2)
		in	3	9(X-2)(Y-2)

積もり方法 1 については，X と Y の値を 100 と指定して求めた結果である．見積もり方法 2 は，X と Y で見積もり結果が表現されており，それぞれ 100 とすれば見積もり方法 1 と同じ結果である．表 4 では，解析対象のループを変えて見積もったデータ転送量と，解析対象内のデータ転送量に対象の外側のループ回数を乗じた filter 関数全体のデータ転送量を表記している．ループ回数は，ループ変数が正規化されているため，ループ変数の範囲から分かる．表 4 から，LLVM/Polly を利用してループの階層ごとにデータ転送量が求められていることが分かる．

5. おわりに

本稿では，コンパイラ基盤 LLVM を用いてデータ転送量を見積もる方法を提案・実装し，画像フィルタプログラムに適用した．この結果，ループの階層ごとにデータ転送量を求められていることを示した．今後は，提案手法を改良し，より複雑なアプリケーションに対してデータ転送量を見積もり，FPGA アクセラレータ設計の支援効果を示す予定である．

参考文献

[1] 富田憲範, 一場利幸, 田宮豊, 藤澤久典, 今里賢一, 山下公彰, “FPGA アクセラレータ開発を支援するためのツール環境”，2017-SLDM-181, デザインガイア 2017.
[2] The LLVM Compiler Infrastructure Project，http://llvm.org/.
[3] J. Zhang, Z. Zhang, S. Zhou, M. Tan, X. Liu, X. Cheng, and J. Cong, “Bit-level optimization for high-level synthesis and FPGA-based acceleration,” in Proc. FPGA 2010.
[4] Polyhedral optimizations for LLVM, https://polly.llvm.org/.
[5] 尾形冬馬, 中野秀洋, 宮内新, “多面体モデルに基づく並列化コンパイラにおける制約条件の緩和”，IEICE-CPSY2014-151, 2014.
[6] Tobias Grosser, “Enabling Polyhedral Optimizations in LLVM,” Diploma Thesis, 2011. https://polly.llvm.org/publications.html
[7] Google Optimization Tools (OR-Tools), https://developers.google.com/optimization/

© 2018 Information Processing Society of Japan