一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
VLD2018-87,CPSY2018-97,RECONF2018-61(2019-01)

データフロー型計算アプリケーション用DMACの高位合成による自動設計

木田　智大† 　川俣　裕一† 　柴田裕一郎† 　佐野健太郎††

† 長崎大学大学院工学研究科　〒852–8521 長崎市文教町 1–14
†† 理化学研究所 計算科学研究センター　〒650–0047 神戸市中央区港島南町 7–1–26
E-mail: †{kida,kawamata,shibata}@pca.cis.nagasaki-u.ac.jp, ††kentaro.sano@riken.jp

あらまし　本論文では，FPGA クラスタ上で計算アプリケーションに応じてデータフローマシンを構築可能な計算機システムにおいて，計算アプリケーションのソフトウェアコードから高位合成によって，データフローマシンとメモリ間のデータ転送制御を担う DMAC を自動設計する手法を提案する．これにより計算アプリケーションに応じて複雑なアクセスパターンにも対応する DMAC を自動設計することができ，アプリケーション開発者の負担を軽減することができる．本稿では，HLS により DMAC を自動生成するための記述について検討するとともに，実装実験を通じて HLS で生成した DMAC の基本性能を明らかにし，本手法の有効性と課題について議論する．

キーワード　DMAC，FPGA，データフローマシン，高位合成

Tomohiro KIDA†, Yuichi KAWAMATA†, Yuichiro SHIBATA†, and Kentaro SANO††

† Graduate School of Engineering, Nagasaki University　1–14 Bunkyo-machi, Nagasaki, 852–8521 Japan
†† RIKEN Center for Computational Science　7–1–26 Minatojimaminamimachi, Kobe Chuo-ku, 650–0047 Japan
E-mail: †{kida,kawamata,shibata}@pca.cis.nagasaki-u.ac.jp, ††kentaro.sano@riken.jp

1. 序　　論

近年の CPU や GPU を使用した並列計算機は半導体微細化技術の頭打ちや消費電力の増大，複雑化する CPU 間の同期，プログラムの性能向上のための最適化に要する時間の増大などの要因から高い性能を得ることが困難になりつつある．しかし，大規模なニューラルネットワークの学習や科学技術シミュレーションなどの用途として，今後も更に低電力かつ高性能な計算機システムが求められる．このような問題に対し，高位合成技術を使用した計算アプリケーションに特化したハードウェアを生成し，FPGA などの回路再構成可能なデバイスに構築する方式がある [1] [2]．近年では，浮動小数点数演算コアを搭載したり，動作周波数を高めるためのパイプラインレジスタを備えた配線アーキテクチャを備えるアーキテクチャが登場するなど，FPGA にも高性能計算用プラットフォームとしての期待が集まっている [3].

アプリケーションのアルゴリズムをそのままハードウェア化し FPGA で効率よく実装する方式としては，処理の並列性を自然に抽出できるデータフローの考え方が従来から広く用いられている [4] [5]．ただし，並列計算機として模索されたデータフローマシンのアーキテクチャでは，各ノードには動的に演算が割り当てられ，データの同期も各演算ノードで行われるものが一般的であるのに対し，FPGA マシンの場合には演算は静的に割り当てられ，データの同期も粗粒度にとられる場合も多い．後者の方式はしばしばストリーム処理とも呼ばれ [6]，データフローマシンとは厳密には区別されるが，本稿ではこれらを含めて広義にデータフローと呼称する．

本研究グループでは多数の FPGA を接続したクラスタ上に，アプリケーションに応じたデータフローマシンを実現する計算機システムを開発している．データフローマシンは，FPGA のハードウェア資源を効率良く演算に割り当てることができ，高い性能を低電力で実現できるが，各演算ノードはメモリアクセス制御機能を持たないのが一般的である．典型的には DMAC (Direct Memory Access Controller) によって外部メモリから読み出されたデータがストリーム状にデータフローグラフに入力され，その出力ストリームが外部メモリに書き戻される．しかし，FPGA マシンにおいては，計算アプリケーションごとにデータフローの形状は異なるため，メモリへのアクセスパターンも異なる．また，基本的には規則的なメモリアクセスを行うアプリケーションであっても，例えば境界条件処理などで部分的に不規則なアクセスを行う必要がある場合もある．このように，柔軟なメモリアクセスを行える DMAC をアプリケーション毎に

設計するのはアプリケーション開発者にとって大きな負担となる．

FPGA におけるアプリケーション開発の生産性向上には，高位合成 (HLS: High Level Synthesis) を導入し記述の抽象度を高めることが効果的である [7] [8]．特にデータパス設計においては，プログラミング言語と同様の記述からデータフローグラフを自然に抽出できる．これと同様に，プログラミン言語によるアプリケーション記述から，データアクセスに関する情報を抽出し，HLS により DMAC を自動生成できれば，アプリケーション毎に特化した DMAC を容易に開発することができる．そこで本稿では，HLS により DMAC を自動生成するための記述について検討するとともに，実装実験を通じて HLS で生成した DMAC の基本性能を明らかにし，本手法の有効性と課題について議論する．

本稿では，まず，2 節で研究の背景と解決すべき問題点を明らかにし，3 節で提案手法を示す．4 節で HLS 記述から DMAC を自動生成する手法の詳細について述べ，5 節で提案手法で生成された DMAC の性能を評価する．最後に 6 節で結論を述べる．

2. 研 究 背 景

2.1 FPGA クラスタ機構

本研究グループが開発している計算機システムの処理フローを図 1 に示す．計算アプリケーションのソフトウェアコードで記述されたプログラムファイルから計算カーネル (データフローマシン) を生成するハードウェアコンパイラを開発しており，計算アルゴリズムを処理する計算カーネルを複数の FPGA で構成された大規模クラスタ上に構築し，計算アプリケーションを実行する．

このハードウェアコンパイラには，東北大学で開発された SPGen (Stream Processing Generator) [9] を利用し，ストリーム処理の数式記述ファイルを入力しすると，対応するハードウェア計算カーネルの IP コアを出力できる．

このシステムは Intel Platform Designer の枠組みで利用可能である．データ転送制御は 汎用的な DMAC である mSGDMA を使用しており，データフローマシンに外部メモリからデータを入力し，また，計算結果をメモリに格納する処理を担当する．

2.2 研究の動機

しかし，現在の設計環境下で実装されたデータフローマシン計算アプリケーションは，メモリの連続番地アクセスのみに対応しており，複雑なメモリアクセスには未対応である．また，開発者はデータの転送情報をあらかじめ DMAC に記述しなければならない．計算アプリケーションを変更すると，ハードウェアコンパイラから生成されるデータフローマシンも異なるため，これに応じてメモリアクセスのサイズやパタン，タイミングも変更する必要がある．また，FPGA クラスタ上のデータフローマシンへの複雑かつ多量なデータ転送情報の DMAC への記述はなおさら困難であり，アプリケーション開発者への負担が大きい．つまり，ソフトウェアコードからハードウェアコンパイラによってデータフローマシン用のデータ転送制御を自動生成する枠組みが求められる．

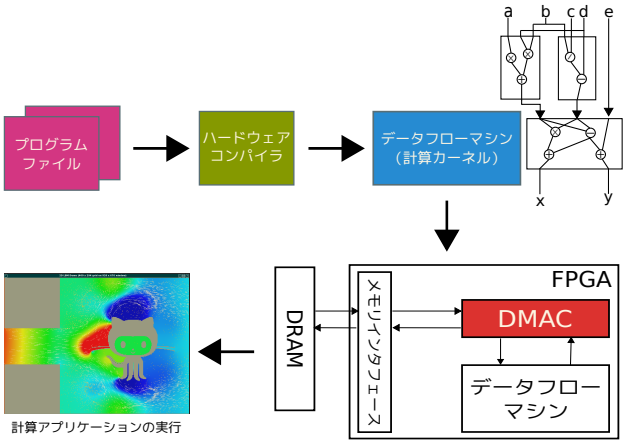

図 1　計算機システムの処理フロー

3. 提 案 手 法

3.1 HLS DMAC の概要

本研究では，上記の課題を解決するために，データ転送制御を担う DMAC を開発者が記述した計算アプリケーションのソフトウェアコードを元に高位合成ツールを使用することによって自動合成する方法を提案する．

その例をコード 1 に示す．配列からデータを取得し，計算し，そのデータを配列に格納する一連の処理である．これは，配列を DRAM，計算処理をカーネルとして考えることができる．この構造を利用して，ソフトウェアコードの配列アクセス記述のみを抽出し，高位合成することで特定の計算アプリケーション用の DMAC を自動合成する．

また，開発するハードウェアコンパイラの処理フローを図 2 に示す．このハードウェアコンパイラはアルゴリズムを記述したソフトウェアコードを入力として，Intel Platform Designer で利用可能なデータフローマシンの計算カーネルコア IP とデータ転送を担う DMAC の IP を出力する．ソフトウェアコードからメモリアクセスと計算処理の分離や SPD，C++ への変換は別稿に譲り [10] [11]，本稿では図 2 の配列アクセスから DMAC IP の生成部分を議論する．

なお，高位合成記述からメモリアクセス機構を自動生成する試みは，これまでにもいくつか研究例がある．例えば PyCoRAM [12] では，Python による高位合成とメモリ抽象化システムを使用し，メモリアクセスパターン記述した Python コードから IP コアとして制御スレッドを生成できる．しかし，外部メモリと直接 DMA 転送を行う DMAC を，アプリケーションに特化した形で高位合成する試みは，これまであまり報告されていない．

3.2 研 究 目 的

本稿では，ソフトウェアコードから抽出した配列アクセス記述から Intel 社の 高位合成ツールである HLS Compiler を使用し，データフローマシン用のデータ転送を制御する DMAC を自動設計する手法について述べる．その実現可能性を示すために，まず HLS により DMAC を自動生成するための記述につい

コード 1　ソフトウェアコードにおける配列アクセス

```
1  void func () {
2    for (int i = 0; i < N; i++) {
3      int tmp1 = a[i]; // read from MEM
4      int tmp2 = b[i]; // read from MEM
5      int tmp3 = (tmp1 + tmp2) / 2; // calculate
6      c[i] = tmp3; // write to MEM
7    }
8  }
```

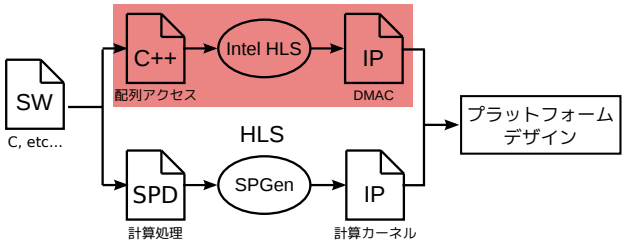

図 2　ハードウェアコンパイラの処理フロー

て検討する．そして，生成されたものが DMAC としてのメモリアクセスの機能や十分な性能を有するかについて，転送サイズやストライドなどのパラメータを変更しながら性能評価実験を行う．

4. HLS DMAC

図 1 にあるように，本システムにおける DMAC は DRAM のメモリインタフェースとデータフローマシンと接続されている．DMAC，メモリインタフェース間は Avalon-MM インタフェースで接続され，DMAC，データフローマシン間は Avalon-ST インタフェースで接続されている．今回は DRAM からデータを読み込み，そのままデータフローマシンにストリーム出力する Read DMAC と，ストリーム入力されたデータを DRAM に書き込む Write DMAC を設計した．

4.1 HLS Read DMAC

引数に CSR (Control Status Register) を用意し，(1) 読み込み処理，(2) 書き込み処理，(3) 動作なし，の 3 種の動作切り替えを可能にしている．

コード 2 に Read DMAC の C++ 記述を示す．4 行目からは Avalon-MM インタフェースとなるクラステンプレートであり，データ幅は 512 ビット，アドレスは 32 ビットとしている．14 行目は生成される Read DMAC のコンポーネント関数である．21 行目がメモリへの書き込みの処理をしており，メモリから取得した値が正しいかどうかを確認するために，あらかじめメモリをデータで上書きしている．26 行目ではメモリのデータを読み込み，ストリーム出力している．

設計したコード 2 を Intel HLS Compiler を使用して IP を出力し，HLS の機能によりリソース消費の見積りを行った．ターゲットデバイスは Arria 10 GX 10AX115N3F45I2SG である．表 1 に生成した Read DMAC の消費リソース量を示す．ALU や FF，RAM，MRAB などのリソースは定量的に少ない量で実装することができる．また，資源の使用割合は RAM が 1 番多く，1.2% だった．

コード 2　Read DMAC の C++ 記述

```
1  #include "HLS/hls.h"
2  #include "HLS/ac_int.h"
3
4  typedef ihc::mm_master<ac_int<512,false>
5                  ,ihc::dwidth<512>
6                  ,ihc::awidth<32>
7                  ,ihc::aspace<1>
8                  ,ihc::latency<0>
9                  ,ihc::maxburst<32>
10                 ,ihc::waitrequest<true>
11                 ,ihc::align<64>
12                 > mm_read;
13
14 component int read_dma (mm_read &mem,
15                  ihc::stream_out< ac_int
                        <512, false> > &
                        data_out,
16                  bool csr_write,
17                  bool csr_read,
18                  int SIZE,
19                  int STRIDE)
20 {
21       if (csr_write && !csr_read) {
22             for (int i = 0; i < SIZE; i++) {
23                   mem[i*STRIDE] = i;
24             }
25             return 1;
26       } else if(!csr_write && csr_read) {
27             for (int i = 0; i < SIZE; i++) {
28                   data_out.write(mem[i*STRIDE
                        ]);
29             }
30             return 2;
31       }
32       return 0;
33 }
```

表 1　Read DMAC の消費リソース量

Flit	ALUTs	FFs	RAMs	DSPs	MLABs
read_dma	2982	6236	34	3	6
available	854400	1708800	2713	1518	12984

4.2 HLS Write DMAC

Read DMAC と同様に CSR を用意し，(1) 書き込み処理と (2) 動作なしの処理を切り替えを可能にしている．コード 3 に Write DMAC の C++ 記述を示す．4 行目は Avalon-MM インタフェースとなるクラステンプレートであり，Read DMAC と同様にデータ幅を 512 ビット，アドレスを 32 ビットとした．14 行目は 生成される Write DMAC のコンポーネント関数である．20 行目でストリーム入力されたデータを DRAM に書き込んでいる．

表 2 に生成した Write DMAC の消費リソース量を示す．Write DMAC も同様にリソースは定量的に少ない．

表 2　Write DMAC の消費リソース量

Flit	ALUTs	FFs	RAMs	DSPs	MLABs
write_dma	1501	7421	18	1.5	1
available	854400	1708800	2713	1518	12984

5. 性 能 評 価

5.1 評 価 方 法

図 3 に評価環境のアーキテクチャを示す．Intel HLS Compiler によって生成された Read，Write DMAC をそれぞれ Intel Platform Designer でメモリインタフェースと接続し，FPGA に

コード 3　Write DMAC の C++ 記述

```
#include "HLS/hls.h"
#include "HLS/ac_int.h"

typedef ihc::mm_master<ac_int<512,false>
                 ,ihc::dwidth<512>
                 ,ihc::awidth<32>
                 ,ihc::aspace<1>
                 ,ihc::latency<0>
                 ,ihc::maxburst<32>
                 ,ihc::waitrequest<true>
                 ,ihc::align<64>
                 > mm_write;

component int write_dma (mm_write &mem,
                  ihc::stream_in< ac_int<512,
                       false> > &data_in,
                  bool csr_write,
                  int SIZE,
                  int STRIDE)
{
      if (csr_write) {
            for (int i = 0; i < SIZE; i++) {
                  mem[i*STRIDE] = data_in.
                       read();
            }
            return 1;
      }
      return 0;
}
```

実装した．そして，DMAC とメモリ間でのデータの転送開始から終了までのサイクル数を計測した．DMAC とメモリインタフェース，計測モジュール区間は 200 MHz のクロック周波数で動作させ，DRAM メモリとメモリインタフェース間は 800 MHz で動作させた．対象とする FPGA は Intel Arria10 GX 10AX115N3F45I2SG，FPGA ボードは 4GB の DDR3L-2133 メモリを 2 枚搭載する TERASIC 社の DE5a-NET を使用し，今回の評価では 1 枚のみ使用した．理論メモリバンド幅は，12.8 GB である．また，検証には Signal Tap Logic Analyzer を使用し，FPGA で実機動作している内部信号をキャプチャした．

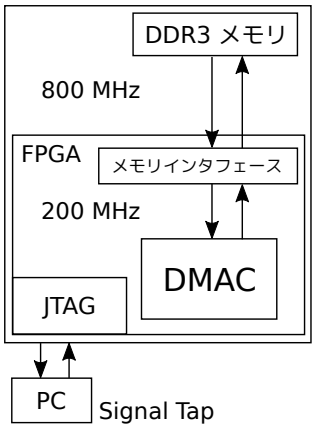

図 3　評価環境のアーキテクチャ

5.2　HLS Read DMAC 結果

図 4 の上図に Read DAMC での DDR からの読み込み開始部分の内部信号のキャプチャ図を示す．転送サイズは 512 個でストライド幅が 8 となっている．バースト数とは DMAC からメモリインタフェースに対して 1 回の転送命令でデータ長 (512 bit) のデータをどのくらい転送 (バースト転送) するかの数である．バースト転送することで 1 回の転送命令でデータ長以上のデータを連続番地からアクセスできる．512-bit (64-Byte) のデータ 1 個は，アドレスの 0x00 から 0x3f までであり，8 個ストライドするので，0x1ff ずつ飛ばしてアクセスされる．メモリインタフェースから取得した Read データをストリームに出力する位置をキャプチャしたものを図 4 の下図に示す．コード 2 にあるように，0 から 511 までを出力している．

ストライド幅とデータサイズを変化させたときの実効メモリバンド幅のグラフを図 5 に示す．データサイズを増加させるにつれ，実効メモリバンド幅も増加し，ストライド幅が 1 のときは理論値に近い値だった．ストライド幅が 128 のときでもメモリバンド幅は約 7.7 GB であり，理論値の 60% だった．

5.3　HLS Write DMAC 結果

ストライド幅とデータサイズを変化させたときの実効メモリバンド幅のグラフを図 6 に示す．

ストライド幅が 1 のときは Read とほぼ同じ性能だったが，データサイズが小さいとき，ストライド幅の増加は Write の実効メモリバンド幅により悪影響を及ぼした．ストライド幅が 128 のとき，データサイズを変更しても，Write の実効メモリバンド幅は低いままだった．

Read の結果と同じように，データサイズを増加させるにつれ，実効メモリバンド幅が増加した．また，ストライド幅が 128 のときでも約 7.6 GB であり，理論値の 60% 程度だった．

Read，Write DMAC ともに，ソフトウェアコードの配列アクセス記述を高位合成することによって，配列アクセスと同じメモリアクセス動作した．実効メモリバンド幅は最低でも理論値の 60% 程度に抑えることができた．しかし，配列でのストライドアクセスは，バースト数が 1 になってしまうため，コード内でプラグマなどの特別な記述をする必要があるだろう．また，今回キャプチャしたデータのバースト数を確認したところ，ストライド幅が 1 のときはバースト数が 32 であり，その他は 1 だった．このことからユーザがバースト数を指定しなくても，配列のアクセスに合わせて自動的にバースト数を上げてくれることが分かる．

6.　結　　論

本研究では，ソフトウェアプログラムからハードウェアコンパイラにより生成し，FPGA に構築されたデータフロー型計算アプリケーションにおいて，データフローマシンと DRAM 間のデータ転送制御モジュール DMAC を高位合成によって生成する方法を提案し，メモリから Read，メモリへの Write をする DMAC を設計した．生成された DMAC はソフトウェアコードの配列アクセス記述と同じようにメモリにアクセスし，データ転送を行った．そして，その DMAC の性能をパラメータを変更したときの動作を検証し，評価した．今後の課題として，ランダムアクセスのような複雑な配列アクセスでの動作検証やデータフローマシンと DMAC との接続することが挙げられる．

文　　献

[1] W. Vanderbauwhede and K. Benkrid, eds., High-Performance Computing Using FPGAs, Springer-Verlag New York, 2013.

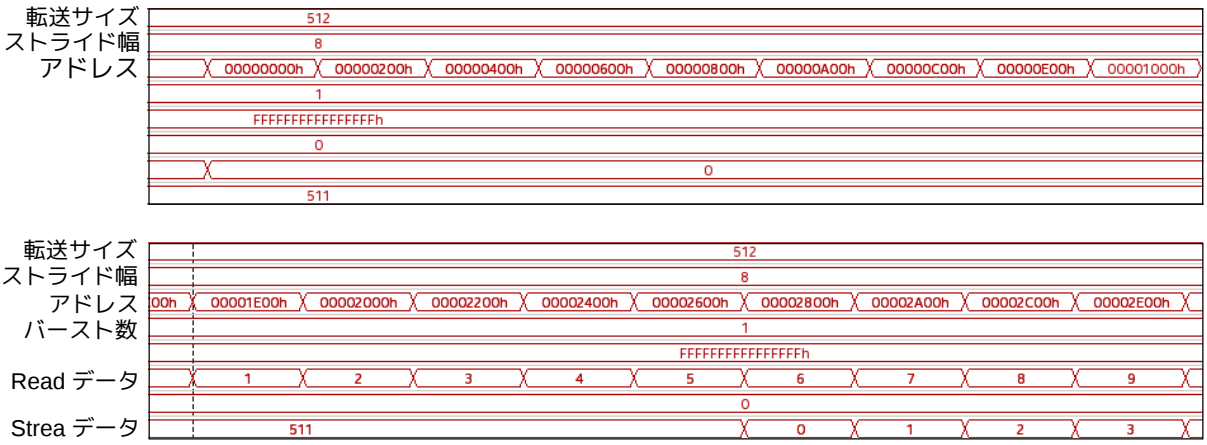

図 4　Signal Tap で見る Read DMAC の 内部信号

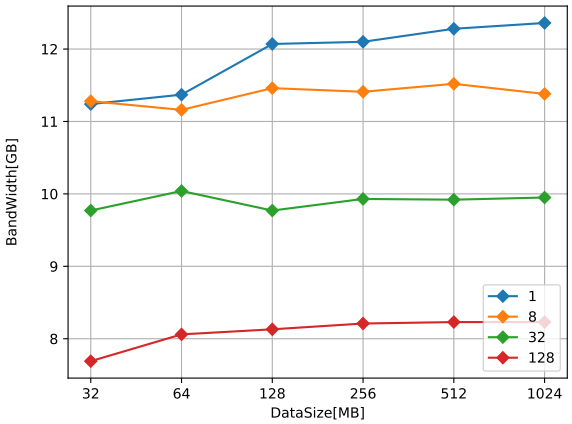

図 5　Read DMAC のデータサイズ，ストライドを変更したときのメモリバンド幅

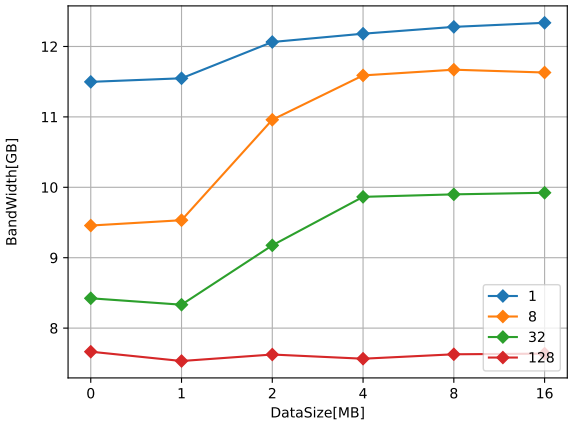

図 6　Write DMAC のデータサイズ，ストライドを変更したときのメモリバンド幅

[2] 長洲航平，佐野健太郎，河野郁也，中里直人，“ストリーム計算ハードウェアコンパイラ SPGen を用いた FPGA 津波シミュレータの開発，” 情報処理学会研究報告：システムと LSI の設計技術（SLDM），vol.2016，no.23，pp.1–6，2016.

[3] K. Sano and S. Yamamoto, “FPGA-based scalable and power-efficient fluid simulation using floating-point DSP blocks,” IEEE Transactions on Parallel and Distributed Systems, vol.28, no.10, pp.2823–2837, 2017.

[4] X.-P. Ling and H. Amano, “WASMII: a data driven computer on a virtual hardware,” Proc. IEEE Workshop on FPGAs for Custom Computing Machines (FCCM), pp.33–42, 1993.

[5] M.J. Flynn, O. Pell, and O. Mencer, “Dataflow supercomputing,” Proc. International Conference on Field-Programmable Logic and Applications (FPL), pp.1–3, 2012.

[6] K. Dohi, K. Okina, R. Soejima, Y. Shibata, and K. Oguri, “Performance modeling of stencil computing on a stream-based FPGA accelerator for efficient design space exploration,” IEICE Transactions on Information and Systems, vol.98–D, no.2, pp.298–308, 2015.

[7] 三好健文，“FPGA 向けの高位合成言語と処理系の研究動向，” コンピュータソフトウェア，vol.30，no.1，pp.76–84，2013.

[8] 渡邊　実，佐野健太郎，高前田伸也，三好健文，中條拓伯，“FPGA ハードウェア・アクセラレーション向け日の丸高位合成ツール，” 電子情報通信学会論文誌 B，vol.J100-B，no.1，pp.1–10，2017.

[9] 伊藤　涼，鈴木隼人，千葉諒太郎，佐野健太郎，山本　悟，“ストリーム計算のための高位合成コンパイラの設計と実装，” 電子情報通信学会技術研究報告：リコンフィギャラブルシステム（RECONF），vol.113，no.418，pp.1–6，2014.

[10] 李　珍泌，上野知洋，佐藤三久，佐野健太郎，“FPGA 向けストリームプロセッサ生成のための C 言語フロントエンドの開発，” 情報処理学会研究報告：ハイパフォーマンスコンピューティング (HPC)，vol.2018，no.25，pp.1–7，2018.

[11] 李　珍泌，上野知洋，佐藤三久，佐野健太郎，“ストリーム計算ハードウェアコンパイラ SPGen のための Polyhedral Model を用いたループスケジュール最適化，” 情報処理学会研究報告：ハイパフォーマンスコンピューティング (HPC)，vol.2018，no.12，pp.1–6，2018.

[12] 高前田伸也，吉瀬謙二，“メモリ抽象化フレームワーク PyCoRAM を用いたソフトプロセッサ混載 FPGA アクセラレータの開発，” 情報処理学会研究報告：計算機アーキテクチャ (ARC)，vol.2014，no.8，pp.1–4，2014.