高位合成を用いた3Dステンシル計算のFPGAアクセラレーション: Maxelerシステムにおけるケーススタディ

柴田 裕一郎[1,a)]

概要：ステンシル計算は，様々な科学技術シミュレーションに頻用される計算パターンのひとつであり，FPGA 上に問題に応じた深いパイプライン構造のハードウェアを構成し，ストリーム処理化することにより高い効果をあげることができる．しかし，そのようなハードウェアを HDL 記述で設計するのは非効率であり，高位合成技術への関心が高まっている．一方で，高位合成によるアプリケーション開発には，従来のソフトウエア開発とは異なる最適化手法も必要となる．本稿では Maxeler Technologies 社が提供する Java ベースの高位合成系を例に，3D ステンシル計算の実装例を示し，その効果と課題を報告する．

FPGA Acceleration of 3D Stencil Computing with High Level Synthesis: A Case Study on Maxeler Systems

Yuichiro Shibata[1,a)]

Abstract: Stencil computation, which is one of frequently used computing patterns in various scientific simulations, can be efficiently executed in a stream processing manner by configuring deep-pipelined application-specific hardware on FPGAs. HDL-based design of such hardware, however, is often a heavy burden for application designers, and thus high level synthesis (HLS) techniques have been receiving increasing attention. On the other hand, HLS design needs different optimization approaches from conventional software design. This paper reports and discusses some case study results of FPGA-based 3-D stencil computation designed with Java-based HLS tools developed by Maxeler Technologies.

1. はじめに

FPGA（Field Programmable Gate Array）は，デバイスアーキテクチャの改良と半導体プロセス技術の発展に支えられ，ますます高性能かつ大規模になっており，その応用分野も広がりを見せている．とりわけ近年では，高い性能電力比を達成可能な計算アクセラレータとして商業的な関心も高まっている．

一方，FPGA におけるアプリケーション開発環境は回路設計ツールを源流として発展しており，VHDL や Verilog HDL などの HDL（Hardware Description Language）を用いて，クロックサイクルごとハードウェアの振る舞いを RTL（Register Transfer Level）で記述するのが主流となっている．IP（Intellectual Property）コアの自動生成機能を用いることにより，設計済みのハードウェアライブラリをカスタマイズして再利用することも可能であるが，適切なパラメータの選択したり，どのクロックサイクルでどのハードウェアを利用するかといったスケジューリングについては，ユーザが決定し明示しなければならない．アプリケーション開発生産性の低さが FPGA のひとつの弱点として認識されている [1]．

このような背景から，C や Java などのプログラミング言語並に抽象度の高い記述から HDL 記述を生成する高位合成ツールへの期待が高まっている．すでに商用パッケージを含む様々な高位合成ツールが利用可能であり [2], [3], [4], [5]，これらを用いることで，ユーザは RTL 設計よりも容易に対象のアプリケーションを記述できる．

[1] 長崎大学大学院工学研究科
Graduate School of Engineering, Nagasaki University, Bunkyo-machi, Nagasaki 852–8521, Japan
[a)] shibata@cis.nagasaki-u.ac.jp

しかし，高位合成による FPGA アプリケーションの開発は，まだ十分な設計経験の蓄積と共有がなされておらず課題も多い．まず，高位合成には，プログラミング言語による逐次的な意味を基本とする記述から並列動作するハードウェアを推論するという難しさがあり，ユーザにとって，記述コードと生成されるハードウェア構成の関係を把握することは容易ではない．例えば，「高位合成向き」の記述の仕方というものが存在するのか，あるいは一般的なソフトウェア記述とは異なる最適化アプローチが必要なのか，あるとすればそれはどういうものかといった問題は自明ではない．また，アプリケーションを FPGA にマッピングする際の論理合成や配置配線に要する時間は，高位合成によっても短縮されることはないし，デバイスの大規模化と共にむしろ増大する傾向にある．したがって，高位合成によって簡単にアーキテクチャレベルの設計パラメータを変更できるものの，最適なアーキテクチャのパラメータ探索には大きな時間コストが必要となり，効率的な設計空間探索手法が必要となる．

このような背景から，著者らのグループでは高位合成による 3 次元ステンシル計算の FPGA 実装を通じて，記述や合成方法によってハードウェアの特性や性能がどのように最適化されるかを評価し，設計空間探索モデルの構築を進めてきた．本稿では，Maxeler Technologies 社が提供する高位合成系と FPGA アクセラレータを用いたケーススタディを示し，現状と今後の課題について議論する．

2. Maxeler システム

MaxCompiler は Maxeler Technologies によって開発が行われている Java ベースの高位合成系である．同社が提供する FPGA アクセラレータへの実装を前提としたハ HPC（High Performance Computing）分野をターゲットアとしたコンパイラであり，ストリーム指向のプログラミングモデルを採用している．ユーザが MaxJava と呼ばれる Java ベースの言語でアプリケーションを記述すると，その記述に基づいてパイプライン構造をもつ計算ハードウェア（カーネル）や，PCIe インタフェース，DRAM インタフェース，FPGA 間通信インタフェース，ホストプログラムからアクセスするための API モジュールなどが生成される．さらに，ユーザアプリケーションの単体テスト環境やデザイン全体のシミュレーション環境も用意されており，ユーザが HDL を一切記述しなくても FPGA アプリケーションを開発することができるようになっている．

MaxCompiler は大規模なパイプライン構造を構成することによって，高スループットな計算処理を実現することを指向している．このため，回路規模削減のためのリソースシェアリング等は自動で行われず，このような処理を望む際にはユーザが明示的に記述する必要がある．

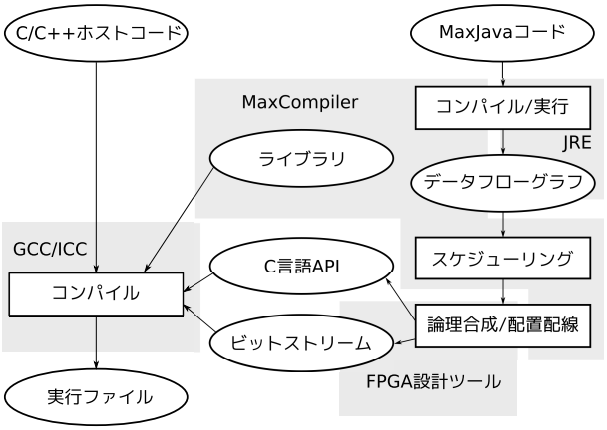


図 1 Maxeler システムにおけるコンパイル環境の概要

2.1 MaxCompiler とデータフロー生成

前述のようにすでに数多くの高位合成システムが提案されているが，これらは概ね以下の 2 種類に大別できる．

(1) 高位合成記述の演算的意味がハードウェアの構造を導くもの
(2) 高位合成記述がハードウェア構造を組み立て方を表すもの

前者のシステムでは，ユーザが記述したプログラムの演算的な意味が直接データフローグラフの形に対応しており，コンパイラはこれを FPGA 演算回路に変換する．このため，ポインタの使用や動的なメモリ確保が禁止されるなど，ユーザが記述できる内容には大きな制約が課せられるため様々な工夫が求められる [6], [7].

一方，MaxCompiler は後者に属し，ユーザ記述の演算意味がハードウェアに対応するのではなく，ハードウェア構造の構築手続きをユーザが記述する方式をとる．データフローの各ノードを表現するためのクラスがコンパイラ環境が提供する MaxJava ライブラリ中で定義されており，そのインスタンス間の接続をユーザがソースコード中で指示することによりデータフローグラフが生成される．最終的なデータフローグラフは，ユーザが記述したソースコードを通常のプロセッサ上で実行した結果として生成される．データフローグラフを構築するにあたり，Java の標準的な文法機能を制約なく活用できるため，コードの保守性を向上できる．また，データフローグラフを構築するための実行時間は生成されるハードウェアの性能に影響しないため，ユーザは計算速度ではなく可読性を重視したコードを記述できるメリットもある．このように，MaxJava によるユーザ記述コードは演算の構造をそのまま表現しているわけではなく，データフローグラフを生成するための手続きを示している．**図 1** に MaxCompiler のコンパイル環境の概要を示す．

図 2 に MaxJava によるユーザ記述例を示す．まず，データフローグラフのノードに対応する `HWVar` クラスのインスタンス `x` には入力ストリーム “x” から与えられる単精度

```
1 HWVar x = io.input("x", hwFloat(8, 24));
2 HWVar y = constant.var(hwFloat(8, 24), 0);
3 if (heavy_user_function())
4   y = x * constant.var(hwFloat(8, 24), 2);
5 else
6   y = x - constant.var(hwFloat(8, 24), 1);
7 io.output("y", y, hwFloat(8, 24));
```

図 2 MaxJava による記述例

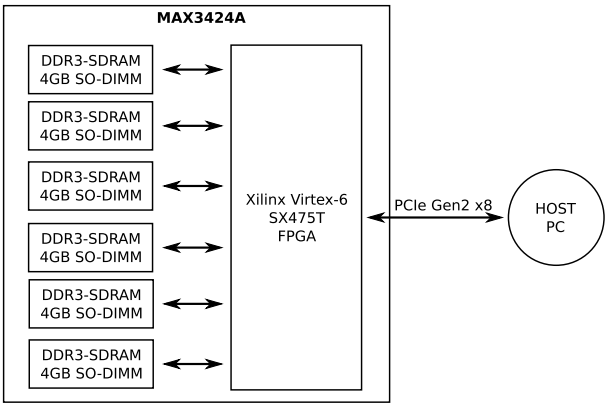


図 3 MAX3424A アーキテクチャ概要

浮動小数点数の値が格納される． その後，3 行目において `heavy_user_function()` 関数の返戻値によって `y` の値を決定するデータフローグラフが変わる．最終的に本カーネルの出力ストリーム “y” には，この `y` の値が出力される．ここで，`heavy_user_function()` の返戻値は静的に決定され，データフローグラフへの入力には影響しないという前提がある．したがって，この関数の実行時間は生成されるハードウェアの性能には影響しない．このような設計思想により，MaxJava においては，標準ライブラリ中のデータ構造やユーザの独自クラスをデータフローグラフの構築に活用できる．

2.2 MAX3424A データフローエンジン

MaxCompiler がターゲットとしている Maxeler 製の FPGA アクセラレータは DFE（Data Flow Engine）と呼ばれる． **図 3** に本稿で示す実験に用いた DFE である MAX3424A の構成を示す． MAX3424A には Xilinx 社の Virtex-6 SX475T FPGA が搭載されている．FPGA には容量 4 GB の DDR3 DRAM モジュールが 6 つ並列に接続されており，38.4 GB/sec のピークメモリバンド幅を提供している．FPGA はホストプロセッサと PCIe gen2 x8 インタフェースで接続されている．

3. MaxGenFD とステンシル計算

MaxGenFD は 3 次元ステンシル計算アプリケーションの開発を支援するために MaxCompiler 上に構築されたフレームワークである [8]．ユーザが MaxCompiler の特徴であるストリーム指向処理をベースとして，容易にコードを記述できるように各種ライブラリを提供している．

3.1 ステンシル計算と領域分割

ステンシル計算は全データに対し，同じ形状のデータ参照（ステンシル）を伴う演算処理を繰り返し適用するデザインパターンの一種である．この計算構造は画像処理における移動ウィンドウ処理と同様であり，ラインバッファを用いたストリーム処理と同様の手法で効率的に FPGA で実現できる [9]．ただし，多次元のステンシル計算では計算領域が大きい場合には，バッファリングしなければならない領域のサイズが肥大化する．そこで，計算領域を小さな領域に分割し，分割した領域ごとに処理することで空間的局所性を増加させるブロック化の手法がしばしば用いられる [10]．なお，隣接する他のブロックの要素を参照する必要があり，この領域は袖領域と呼ばれる．

MaxGenFD の環境では，計算領域のサイズはユーザが実行時に指定する．ユーザにより与えられた計算領域は，まずブロックと呼ばれる単位で分割される．ブロックによる分割は X 次元方向と Y 次元方向だけについて行われる．すなわち，計算領域 (X, Y, Z) とブロックサイズ (B_W, B_H) が与えられると，計算領域は (B_W, B_H, Z) の領域に分割され処理される．

FPGA 上でのストリーム処理と外部 DRAM へのアクセスでは，望ましいデータアクセスパターンが異なる．ストリーム処理ではアクセスの時間的局所性を高めることでバッファリングに必要な BRAM の容量を節約するのが望ましいのに対して，DRAM ではなるべく連続データを転送するのがよい．そこで，MaxGenFD では各々のブロックをさらにタイルと呼ばれる (T_W, T_H) の 2 次元配列に分割し，外部 DRAM へのアクセスはこのタイルを単位に行われるようになっている．メモリから読み出されたタイルは FPGA 内部の BRAM でバッファリングされ，必要に応じて演算パイプラインに順次与えられる．ブロックとタイルによる 2 段階の分割により，ストリーム処理のバッファリングに適した分割法と，DRAM のアクセスに適した分割法のギャップを解消している．なお，ブロックサイズ (B_W, B_H) とタイルサイズ (T_W, T_H) はユーザが合成時に決定する．

3.2 マルチパイプラインとマルチステップ

3 次元ステンシル計算の高速化手法には，演算パイプラインを並列に並べる方法と，各格子データの更新を複数回行ってからメモリに書き戻す方法がある．本稿では，それぞれマルチパイプラインとマルチステップと呼ぶ．

図 4 (a) はステンシル計算のパイプライン構造を模式的に示したものである．マルチパイプラインでは，図 4 (b) に示すように演算パイプラインを空間的並列化し，複数の格子要素を同時に更新する．パイプライン間でバッファリ

ング領域を共有できるため，BRAM の増加は小さいが，外部 DRAM への要求メモリバンド幅の増加は大きい．メモリバンド幅が十分に広ければパイプラインの数に比例して性能が向上する．なお，MaxGenFD では，パイプライン数はタイルの幅 T_W を割り切る数でなければならない．

マルチステップは，図 4 (c) に示すように演算パイプラインを直列に接続し，メモリアクセスあたりの要素更新回数を増やす手法である．マルチパイプラインとは対照的に，DRAM への要求メモリバンド幅を増やすことなく演算性能を高めることができるが，バッファリングに必要な BRAM 容量がほぼステップ数に比例して増加する．指定可能なステップ数については特にマルチパイプラインのような制約はなく，1 ステップ単位で設定できる．

図 4 (d) に例示するように，マルチパイプラインとマルチステップは併用可能である．FPGA の資源量やアプリケーションの特性に応じてマルチパイプライン数やマルチステップ数を適切に設計することが重要となる．

4. 資源性能モデル

マルチパイプライン数とマルチステップ数は MaxCompiler 上で数字を指定するだけで容易に変更することができる．しかし，1 つのデザインを FPGA にマッピングするには論理合成や配置配線などの時間的なコストの高い処理が必要である．このため，FPGA の性能を引き出すためには，要求資源量や性能を予測するモデルを構築し，事前に設計空間探索の範囲を絞り込むことが重要となる [11].

4.1 資源制約モデル

まず，FPGA における演算ハードウェアの要である DSP モジュールについて考える．マルチパイプライン数を N_p，マルチステップ数を N_s，単体のパイプラインに要する DSP モジュールの数を $N_{\mathrm{DSP}}^{(1,1)}$，ターゲット FPGA で利用可能な DSP モジュール数を A_{DSP} とすると，DSP による資源制約は，

$$N_p \times N_s \times N_{\mathrm{DSP}}^{(1,1)} \leq A_{\mathrm{DSP}} - \alpha \tag{1}$$

と表せる．ここで，α は演算パイプライン以外で使用される DSP モジュールの数である．BRAM の資源制約は，

$$\left(N_{\mathrm{BRAM}}^{(1,1)} + C_p \times N_p\right) \times N_s \leq A_{\mathrm{BRAM}} - \beta \tag{2}$$

の表せる．ここで，$N_{\mathrm{BRAM}}^{(1,1)}$ は単体のパイプラインに要する BRAM の使用量，C_p は N_p に対する係数，A_{BRAM} は利用可能な BRAM の総数，β は演算パイプライン以外で使用される BRAM 数である．式 (1)，(2) より，選択可能な N_p と N_s の組合せが得られる．

4.2 演算性能モデル

次に外部メモリのバンド幅を考慮しないときの性能を考える．計算領域のサイズを X，Y，Z，ブロック分割のサイズを B_W，B_H，タイル分割サイズを T_H，T_W，演算パイプラインのクロック周波数を F_s とすると，ピーク演算性能は，

$$P_{\mathrm{compute}} = \eta N_p N_s F_s \,[\mathrm{elements/sec}] \tag{3}$$

と表せる．ここで，η は袖領域へのアクセスの影響をモデル化する演算効率であり，1 ブロックの計算で外部メモリから読み出される要素数

$$G_{\mathrm{read}} = \left(B_W + 2\left\lceil\frac{S_d N_s}{T_W}\right\rceil T_W\right)\left(B_H + 2\left\lceil\frac{S_d N_s}{T_H}\right\rceil T_H\right) Z \tag{4}$$

と 1 ブロックの計算で計算され外部メモリに書き込まれる要素数 $G_{\mathrm{write}} = B_W \times B_H \times Z$ を用いて

$$\eta = \frac{G_{\mathrm{write}}}{G_{\mathrm{read}}} \tag{5}$$

と表される．ここで，S_d はステンシルのサイズである．

4.3 メモリ制約性能モデル

外部メモリへのバンド幅によって制約される際の性能は，

$$P_{\mathrm{mem}} = G_{\mathrm{write}} \frac{T_{\mathrm{mem}}}{B_{\mathrm{mem}}} N_s \,[\mathrm{elements/sec}] \tag{6}$$

と表せる．ここで，T_{mem} はピークメモリバンド幅，B_{mem} は 1 ブロックの計算に必要な転送データ量を示している．N_{mem} をメモリのチャネル数，F_{mem} をメモリバスクロック周波数とすると，

$$T_{\mathrm{mem}} = 8 \times N_{\mathrm{mem}} \times 2 \times F_{\mathrm{mem}} \,[\mathrm{Bytes/sec}] \tag{7}$$

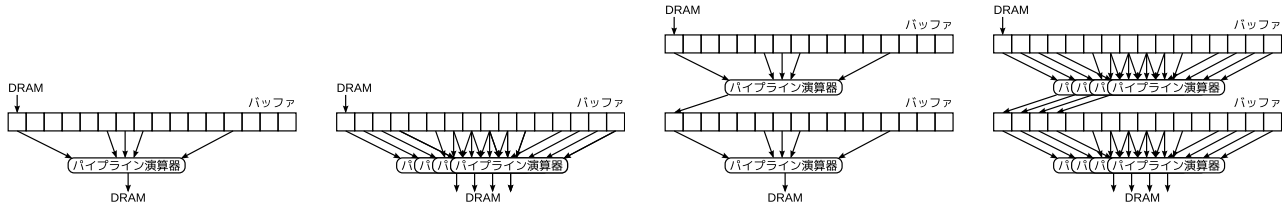
となる．B_{mem} はアプリケーションの演算構成に依存する．MaxGenFD はデータストリームの圧縮機能も持っているため，

$$B_{\mathrm{mem}} = G_{\mathrm{write}} \sum_{i=1}^{n} \gamma_i W_i \,[\mathrm{Bytes}] \tag{8}$$

と表す．ここで，i 番目のストリームの γ_i と W_i は圧縮率とデータ幅である．厳密に γ_i の値を知ることは難しいが，MaxGenFD で設定する圧縮率の目標値を用いることができる．式 (3)，(6) より，ピーク性能は

$$P = \min\left(P_{\mathrm{compute}}, P_{\mathrm{mem}}\right) \tag{9}$$

と表せ，設計探索は式 (1)，(2) の制約をみたす N_p と N_s の組合せのうち，式 (9) の P を最大にするものを求める問題として定式化される．



(a) 基本構成　(b) マルチパイプライン数 4　(c) マルチステップ数 2　(d) 併用構成

図 4　マルチパイプラインとマルチステップのパイプライン構成

5. 実装実験と議論

5.1 資源性能モデルの評価

まず，前節で述べた資源性能見積りモデルの有効性を評価するため，陽解法による 3 次元熱拡散シミュレーションを例に実装実験を行った [11]．取り得るすべての N_p と N_s の組について評価した結果，いずれのデザインでも性能見積りモデルは実測性能とよい一致を見せた．一方，資源量予測の結果実装可能と判断されたデザインでも，実際には配置配線ができなかったものが一部あった．配線可能性のモデル化については，今後の重要な課題のひとつである．

FPGA の性能を最大限に引き出すためには，DSP ブロックや BRAM などのハードウェア資源をできるだけ余さず利用することが望ましい．しかし資源使用率を高めると，タイミング制約を満たしつつ配置配線することが困難になり処理時間が増加するというジレンマがある．今回の例ではマッピングに 40 時間近くかかったものがあった．したがって，最適な N_p と N_s の組合せを網羅的に探索することは現実的ではない．また，N_p と N_s の設定値がもたらす最大の性能差は，浮動小数点数演算バージョンで 20 倍，固定小数点数バージョンでは 60 倍を上回り，資源性能モデルに基づいて配置配線を行うことなく最適なパラメータを選択することが，生産性向上に大きな効果を持つことが分かった．

5.2 電力性能比の評価

次に電力性能比を最適化するにはどのような設計アプローチをとるべきかについて考える．FPGA アクセラレータへの電源供給ラインにシャント抵抗を挿入し，さまざまな N_p と N_s の組合せに対するデザインの消費電力を実測する実験を行った [12]．また，FPGA 設計ツールによる消費電力のプロファイリングと比較した．

FPGA で消費される電力の内訳をみると，今回の例では DSP ブロックによる消費が支配的であることが分かった．高性能なデザインは消費電力も大きくなる傾向にあるが，電力性能比の最も高いデザインは最も性能の高いデザインであった．すなわち，少なくとも今回評価した熱拡散シミュレーションでは，電力性能比の最適化は性能の最適化と矛盾しないことが明かになった．また，性能が同程度のデザインであれば，マルチステップ数 N_s よりもマルチパイプライン数 N_p を増やした方が，電力性能比の観点では有利であることが分かった．これはマルチパイプラインに比べてマルチステップでは BRAM の利用率が高まり消費電力が増えるためだと考えられる．一方で，クロック周波数の設定を変更させることで，性能と消費電力のトレードオフをとることも考えられ，その効果のさらなる解析は今後の課題である．

5.3 FPGA アクセラレータ特有の最適化

高位合成を用いたアプリケーション設計は従来のソフトウェア設計と異なる視点が必要かという問題にアプローチするため，外部メモリアクセスのプロファイル機能を持たせたフレームワークを実装し，FDTD（Finite-Difference Time-Domain）法に基づく電磁界シミュレーションの評価を行った [13]．電磁界シミュレーションでは電界と磁界の格子データの各次元のデータをストリームとして扱うため，ストリーム数が増加しアプリケーションの構造がより複雑になる．実装の結果，パイプライン数 N_p を増やすことで性能の向上が見られたが，N_p が 4 になると配置配線時にタイミング制約を満たすことができなかった．しかし，使用資源量や外部メモリバンド幅には十分に余裕があった．

高位合成による設計では，演算パイプラインについては基本的に目標のクロック周波数にあわせて演算器のパイプライン化やスケジューリングが自動的に行われるため，配置配線時にタイミング制約が問題になることは稀である．このため，多くのストリームを処理することにより，メモリコントローラが複雑化することが原因と考えた．そこでストリームの数を削減するために，ソースコードにおけるデータ記述を配列の構造体（SoA）形式から構造体の配列（AoS）形式に変更した．この結果，N_p を 8 まで高めることが可能となり，約 3 倍の性能向上を得ることができた．

FDTD 法によるシミュレーションでは電界，磁界の各次元ごとに異なる形状のステンシルを持つ．しかし，AoS 形式のデータ構造を用いるとステンシルが共通化されるため，約 30%の不要なメモリアクセスを生じる．したがって，AoS 化は従来のソフトウェア設計にはなじまない FPGA アクセラレータ特有の最適化であり，設計者には一定程度アーキテクチャ的概念が必要であることを示唆している．

5.4 不規則な計算構造への対応

ステンシル計算による科学技術計算では，格子の空間解像度を上げるによってシミュレーションの高精細化が期待できるが，それに伴い計算コストとメモリ使用量が増加する．この解決法の 1 つに，解適合格子法の一種である BCM（Building Cube Method）がある [14]．BCM では計算領域を異なる大きさの直交格子（キューブ）で分割するが，隣接キューブ間の大きさの比率に制約を設けることで並列計算との親和性を高めている．キューブの粒度はシミュレーションモデルに応じて変更することができる．計算精度に大きく影響する部分ではキューブを細かく分割し，計算精度に大きく影響しない部分では粗いキューブを使用することで，計算精度を保ちつつ計算量を削減できる．しかし，不均一なキューブの使用により計算構造は不規則になり，MaxGenFD の環境でそのまま実装することはできない．

そこで，キューブのサイズ情報や，隣接キューブとのサイズ比率の情報もストリームとして演算パイプラインに供給し，異なるサイズのキューブ間でのデータの補完処理もサポートした新しいフレームワークを Java のクラスライブラリとして実装した [15]．この結果，異なるサイズのキューブ間でのデータ授受のために BRAM の使用率が最大で 54.2%増加したが，サイズ変換に伴う補完処理による演算スループットのロスは 2.7%程度であることが分かり，解適合格子により計算量を削減できることを考慮すると，十分許容可能なオーバヘッドに収まることが分かった．

6. おわりに

本稿では高位合成を用いた 3D ステンシル計算の FPGA アクセラレーションについて，Maxeler のシステムを例に実装実験の結果とそこから得られた知見について述べた．資源性能予測モデルの構築が，FPGA の性能を最大限に引き出す上で有効かつ重要であることが分かった．今後の課題には，配線可能性を考慮した資源モデルの構築，電力性能比を最適化するクロック周波数決定法の確立，より不規則な演算構造を持つアプリケーションの実装経験の蓄積などがある．性能最適化には従来のソフトウェア設計とは異なるアプローチが必要であることも示された．これをどのように体系化してユーザに共有するかも重要な課題であろう．また，東工大の佐藤は Polyhedral モデルによるループタイリング最適化を Maxeler システムに適用して効果をあげており [16]，高位合成技術と自動チューニング技術との融合も今後多いに期待される．

謝辞 本稿は長崎大学工学研究科における土肥 慶亮 博士，中村 芳大 氏，福本 航太 氏，翁 浩二 氏，副島 梨恵 氏らとの共同研究の成果に基づく．小栗 清 名誉教授には多くの有益なご助言を賜った．Maxeler Technologies 社には University Program の支援を受けた．ここに深謝する．

参考文献

[1] Nelson, B.: FPGA Design Productivity — A Discussion of the State of the Art and a Research Agenda, *Proc. International Workshop on Applied Reconfigurable Computing (ARC)*, p. 1 (2009).
[2] Xilinx Inc.: Vivado HSL Design, `http://www.xilinx.com/products/design-tools/vivado/integration/esl-design/index.htm`.
[3] Maxeler Technologies: MaxCompiler, `http://www.maxeler.com/`.
[4] Gao, S. and Chritz, J.: Characterization of OpenCL on a Scalable FPGA Architecture, *Proc. International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pp. 1–6 (2014).
[5] 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).
[6] Miyata, M., Shibata, Y. and Oguri, K.: An Optimization Method Focusing on Fixed-point Arithmetic in Applications for Dynamically Reconfigurable Processor, *Systems and Computers in Japan*, Vol. 38, No. 14, pp. 20–28 (2007).
[7] 志田さや香，土肥慶亮，柴田裕一郎，濱田 剛，正田備也，小栗 清：リコンフィギャラブルマシンにおける DMA 転送とデータ配置の自動最適化手法の検討，電子情報通信学会論文誌 D，Vol. 92, No. 12, pp. 2127–2136 (2009).
[8] Maxeler Technologies: MaxGenFD Tutorial Version 2013.3 (2013).
[9] Sano, K.: FPGA-Based Systolic Computational-Memory Array for Scalable Stencil Computations, *High-Performance Computing Using FPGAs*, Springer, pp. 279–303 (2013).
[10] Datta, K., Murphy, M., Volkov, V., Williams, S., Carter, J., Oliker, L., Patterson, D., Shalf, J. and Yelick, K.: Stencil Computation Optimization and Auto-tuning on State-of-the-art Multicore Architectures, *Proc. Conference on Supercomputing (SC)*, pp. 4:1–4:12 (2008).
[11] Dohi, K., Okina, K., Soejima, R., Shibata, Y. and Oguri, K.: Performance Modeling of Stencil Computing on a Stream-Based FPGA Accelerator for Efficient Design Space Exploration, *IEICE Transactions on Information and Systems*, Vol. E98-D, No. 2, pp. 298–308 (2015).
[12] Okina, K., Soejima, R., Fukumoto, K., Shibata, Y. and Oguri, K.: Power Performance Profiling of 3-D Stencil Computation on an FPGA Accelerator for Efficient Pipeline Optimization, *SIGARCH Computer Architecture News*, Vol. 43, No. 4, pp. 9–14 (2015).
[13] Soejima, R., Okina, K., Dohi, K., Shibata, Y. and Oguri, K.: A Memory Profiling Framework for Stencil Computation on an FPGA Accelerator with High Level Synthesis, *SIGARCH Computer Architecture News*, Vol. 42, No. 4, pp. 69–74 (2014).
[14] Ishida, T., Takahashi, S. and Nakasashi, K.: Efficient and Robust Cartesian Mesh Generation for Building-Cube Method, *Journal of Computational Science and Technology*, Vol. 2, No. 4, pp. 435–446 (2008).
[15] Soejima, R., Shibata, Y. and Oguri, K.: HLS-Based FPGA Acceleration of Building-Cube Stencil Computation, *Proc. International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS)*, pp. 463–474 (2017).
[16] 佐藤幸紀：FPGA アクセラレータにおけるデータ参照局所性の高位最適化，電子情報通信学会技術報告（RECONF），Vol. 117, No. 46, pp. 69–73 (2017).