Vol.2020-ARC-241 No.4
2020/7/30

オイラー動画像誇張処理を対象とした Halide を用いた FPGA 加速実行の設計と実装評価

上野 麟[1,a)] 谷本 輝夫[1,b)] 後藤 孝行[1,c)] 丸岡 晃[2,d)] 川上 哲志[1,e)] 小野 貴継[1,f)] 飯塚 拓郎[3,g)] 井上 弘士[1,h)]

概要：視覚で知覚できない情報を可視化する手法として，オイラー動画像誇張処理（Eulerian Video Magnification）がある．これは肉眼では認識できない微小変化を含んだ動画を入力とし，肉眼で認識できるように増幅加工を施した動画を出力するアプリケーションである．このアプリケーションは画像処理によって構成されるため，処理を施す画像サイズやフレームレートを大きくするに従い，計算量が増大しリアルタイム処理実現に必要な性能制約を満たせない問題が生じる．そこで本研究では，画像処理向けドメイン特化言語である Halide を用いた EVM アプリケーションの FPGA (Field Programmable Gate Array) 実装を行った．その結果，FPGA 加速実行の適用により，ソフトウェア処理時に比べフレームレートは 1.97 倍となった．

1. はじめに

物体の時間的な微小変化を検出することにより，我々はより多くの有益な情報を得ることができる．例えば，微小な肺の動きに基づく呼吸状態の観測や，顔色の変化による脈拍数の推定などが挙げられる [7]．しかしながら，人間の目の時間分解能は 20 ms から 30 ms 程度 [3] であるため，これら物理世界における微小変化を肉眼で捉えることは難しい．そこで，撮影された物体を対象にフレーム間での変化を検出・拡張し，動画像データの再構成により可視化するオイラー動画像誇張処理 [10] が考案された．

しかしながら，フレーム内全画素に対する複数回の画像フィルタ処理が必要となるため，リアルタイム性を満たすことが難しいといった問題がある．実際，現状では撮影と同時に微小変化を拡張・可視化するリアルタイム処理は実現できておらず，その高速化が求められている．

そこで本研究では，書換え可能なハードウェアの一種である FPGA (Field-Programable Gate Array) を利用したハードウェアアクセラレーションによりオイラー動画像誇張処理を高速化する．性能目標値はカメラのフレームレートである 30 fps である．ここで，最も注意すべきことは適切なハードウェア－ソフトウェア分割を実施することにある [9]．実装対象となるアプリケーションに対し FPGA が十分なハードウェア資源を有する場合は完全ハードウェア化が可能である．しかしながら，多くの場合で FPGA のハードウェア資源が不足するため，ハードウェアとソフトウェアのハイブリッド実装となる．本研究で対象とするオイラー動画像誇張処理も大規模なソフトウェアであり，ハードウェア－ソフトウェア分割の探索が高性能化を実現するためのポイントとなる．そこで本研究では，Halide コンパイラ FPGA バックエンドを用いた FPGA（ハードウェア）と汎用プロセッサ（ソフトウェア）のハイブリッド実装を行った．Halide プログラム記述を入力エントリ・ポイントとしたハードウェア－ソフトウェア・ハイブリッド設計が可能となるため，効率的に様々な分割を探索することができる．評価を行った結果，OpenCV 実装によるソフトウェア処理時に比べ 1.97 倍のフレームレート向上を実現した．

本稿の構成は以下の通りである．第 2 章ならびに第 3 章では，オイラー動画像誇張処理，ならびに，ドメイン特化言語 Halide と Halide コンパイラ FPGA バックエンドの詳細をそれぞれ説明する．第 4 章では Halide を用いたオイラー動画像誇張処理の実装を示し，第 5 章で FPGA 実装による性能評価を行う．最後に，第 6 章で本研究をまとめる．

[1] 九州大学
[2] 株式会社フィックスターズ
[3] Fixstars Solutions Inc.
[a)] rin.ueno@cpc.ait.kyushu-u.ac.jp
[b)] tteruo@kyudai.jp
[c)] takayuki.goto@cpc.ait.kyushu-u.ac.jp
[d)] akira.maruoka@fixstars.com
[e)] satoshi.kawakami@cpc.ait.kyushu-u.ac.jp
[f)] takatsugu.ono@cpc.ait.kyushu-u.ac.jp
[g)] t_iizuka@fixstars.com
[h)] inoue@ait.kyushu-u.ac.jp

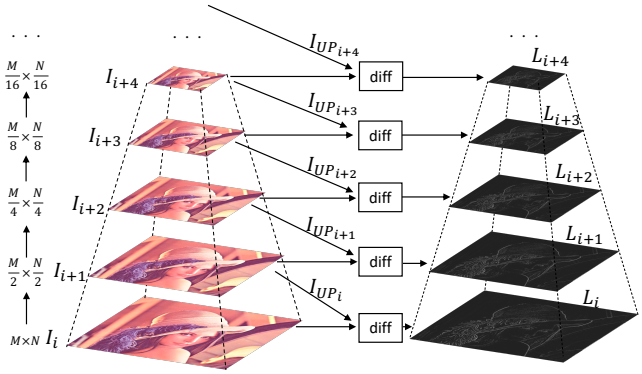


図 1 ラプラシアンピラミッドの処理イメージ図

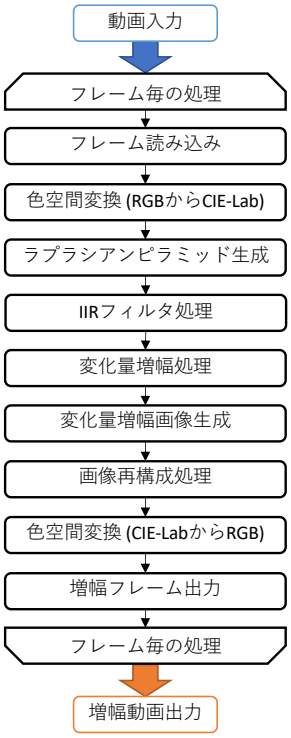


図 2 CIE-Lab 色空間におけるオイラー動画像誇張処理の処理フロー

2. オイラー動画像誇張処理

2.1 概要

オイラー動画像誇張処理は Hao-Yu らによって提案された動画中に含まれた微小変化を増幅する加工手法である [10]．その処理は空間的処理部と時間的処理部の 2 つの重要な処理によって構成される．本研究で対象としたオイラー動画像誇張処理は動画から切り出した 1 フレームに対して空間的処理部でラプラシアンピラミッドを生成し，時間的処理部で IIR (Infinite Impulse Response) フィルタ処理を行う．

2.2 ラプラシアンピラミッド

空間的処理部ではラプラシアンピラミッド [2] を用いる．ラプラシアンピラミッドとは画像ピラミッド [1] と呼ばれる加工手法の 1 つであり，ビジュアルアプリケーションでしばしば利用される．図 1 にラプラシアンフィルタの処理イメージを示す．アルゴリズムは以下の通りである．

(1) 画像 I_i に対して 5×5 のガウシアンフィルタ [8] を用いてコンボリューションを行った結果を I_{Gi} とする．

(2) I_{Gi} の左端上を基点とし，偶数行列番目に位置する画素を全て削除してダウンサンプリングを行い I_{i+1} とする．I_{i+1} は元画像 I_i の $\frac{1}{4}$ 倍の面積となる．

(3) 画像 I_{i+1} の行と列毎に画素値 0 で埋めて面積を画像 I_i と同サイズに拡大し，画像に対してガウシアンフィルタでコンボリューションを行いアップサンプリングした結果を I_{UP_i} とする．

(4) $I_i - I_{UP_i}$ の計算結果より L_i を求める．

以上の処理を複数回繰り返すことによりラプラシアンピラミッドを得ることができる．この空間的処理により画像に含まれる画素値の変化の大きい物体境界を抽出することができる．

2.3 IIR フィルタ

時間的処理部では IIR フィルタを利用する．無限インパルス応答システムというある信号をシステムに入力した際の影響を無限先時間の出力にまで与える性質をもったシステムがある．このようなシステムをフィルタとして利用するため IIR フィルタと呼ばれる．式 (1) は差分方程式を IIR フィルタと見做した式の一例である．

$$y[n] = a\sum_{j}^{A} x[n-j] - b\sum_{j+1}^{B} y[n-j] \qquad (1)$$

n は時刻，$y[n]$ は出力画像，$x[n]$ は入力画像であり，a，b はそれぞれ現在の入力画像に与えられるフィルタ係数と過去の出力画像に与えられるフィルタ係数を表す．A は現在の入力画像に与えられる次数，B は過去の出力画像に与えられる次数であり，定数 1 である．ユーザは式 (1) のフィルタ係数にあたる高周波数帯と低周波数帯のそれぞれでカットオフ周波数を設定することによって通過帯域幅を決め IIR フィルタの設計を行う．ユーザは式 (1) のフィルタ係数 a，b を適切に設定することでラプラシアンピラミッドで抽出した物体境界から微小変化増幅に必要な成分のみを取り出すことができる．

2.4 オイラー動画像誇張処理のフロー

オイラー動画像誇張処理を構成する機能のフローを図 2 に示す．オイラー動画像誇張処理は入力動画から画像フレームを 1 枚切り出し，フレーム毎に処理をして加工を行う．処理されたフレームを動画化することで微小変化を増

幅した動画を生成する．

処理工程を説明する．まずはじめに，切り取られた画像フレームの色空間を RGB 色空間から CIE-Lab 色空間 [4] へ変換を行う．次に，ラプラシアンピラミッドの生成を行う．この時に解像度の異なる画像フレームが構築される．そしてピラミッドの各段の画像に IIR フィルタ処理を施す．この処理により増幅すべき微小変化の抽出を行う．増幅係数をかけることで抽出した微小変化を視覚的に大きく誇張する加工を施す．その後，ピラミッド状態の画像から拡大と画像の足し合わせを行い再び 1 枚の画像を生成する．最後にここまでの処理結果と CIE-Lab 色空間変換結果を足し合わせ，RGB 色空間に変換し直す．この一連の処理をフレーム毎に行い，動画化することで微小変化を誇張した動画を生成できる．

本来，オイラー動画像誇張処理は微小色変化に対しても誇張処理を可能であるが，本研究ではラプラシアンピラミッド生成による物体境界を抽出する特性上，微小な物体境界変化が含まれた動画を入力する必要がある．

3. Halide を用いた FPGA 加速実行について

3.1 ドメイン特化言語 Halide

Halide は 画像処理に特化したプログラミング言語である．Halide は関数の引数に同じ入力値を与えた場合，出力値は常に同じ値を返すという参照透過性を持つため，変数の再代入を制限する純関数型言語の特徴を有する．また，アルゴリズム部とスケジューリング部を分離した記述ができるため，アルゴリズム部で記述した数学的性質を保った上で，スケジューリング部で高速化手法やアーキテクチャ専用最適化の適用が可能となる．そのため可読性の高さと画像処理の高速実行を両立させることが可能である．

しかしながら，Halide はドメイン特化言語であり，汎用的なプログラミング構文記述ができない．そこで Halide は手続き型言語である C++ をラップする形で提供され，ドメイン特化な処理と汎用的プログラムによる処理を組み合わせたフレキシブルな記述を可能にする．

3.2 Halide コンパイラ FPGA バックエンドを利用した FPGA 設計

Halide コンパイラ FPGA バックエンドは株式会社フィックスターズによって開発されている Halide コンパイラである [5]．これは, Halide プログラムを入力とし, Xilinx 社の高位合成処理系である Vivado HLS (High-level Synthesis) [11] 向けに最適化された C++ プログラムを生成するツールである．`unroll` などのスケジューリング記述を HLS 向けに解釈してコード生成する他，`hls::stream` のようなストリーム I/O を利用するためのインターフェース指定など，ハードウェア生成向けのスケジューリング拡張がなされている．

HLS により生成された IP (Intelligent Property) は Vivado ツールによりブロックデザインを行い，論理合成および配置配線することでターゲット FPGA 向けのビットストリーム生成に利用できる．本研究における IP 生成では，画像入出力は全てストリーム I/O となるよう Halide 記述を作成した．この場合，arm プロセッサに接続されている主記憶からデータを read/write するための DMA (Direct Memory Access) エンジンが必要となるが，これらは Xilinx 提供の AXI DMA Controller を用いた．

画像サイズ指定のためのインターフェースは UIO (Universal I/O) デバイスとして認識されるハードウェアモジュールのレジスタとして提供される．したがって，ユーザはレジスタに対して PIO (Programmed Input/Output) アクセスすることで画像サイズを設定可能となり，同様のインターフェースは処理のフラグや係数などのパラメータを指定するためにも用いることができる．一方で，FPGA のリソースを利用するバッファのサイズは，高位合成時に確定されている必要がある．そこで，Halide コンパイラ FPGA バックエンドでは入出力画像サイズの取り得る範囲の制約を付与するための `set_extent_interval` API が提供されている．これを用いることで，付与された入出力画像サイズの制約のもとで必要十分なサイズのバッファを確保するコードが生成される．

本研究で加速実行する処理の中には，異なる画像サイズに対して同様の処理を行ったり，画像サイズによって異なる係数を利用した計算を必要とするものがある．それらの処理ごとに回路を FPGA に実装すると多くの回路資源が必要となる．そのため，可変長インターフェースの活用や係数のパラメータ化による柔軟な Halide 記述によって回路を共通化し，回路資源制約を満たす方針を取る．

4. オイラー動画像誇張処理の Halide を用いたハードウェア設計と実装

4.1 ハードウェア–ソフトウェア協調実装

汎用プロセッサを用いたソフトウェア処理と FPGA によるハードウェア処理に分割して構成されたシステムはハードウェア–ソフトウェアパーティショニング [9] に基づいたシステムである．本研究ではこれに基づき，オイラー動画像誇張処理を構成する各機能を Halide コンパイラ FPGA バックエンドを用いてハードウェア化し，ソフトウェア処理と組み合わせた設計を行う．オイラー動画像誇張処理では画像フレーム加工のために，異なるサイズの画像に対し，同じ計算アルゴリズムを複数回行う必要がある．しかしながら，入力データサイズが固定された回路による実装は，回路資源効率の観点から望ましくない．そこで，Halide コンパイラ FPGA バックエンドの可変長対応 IP 生成のため

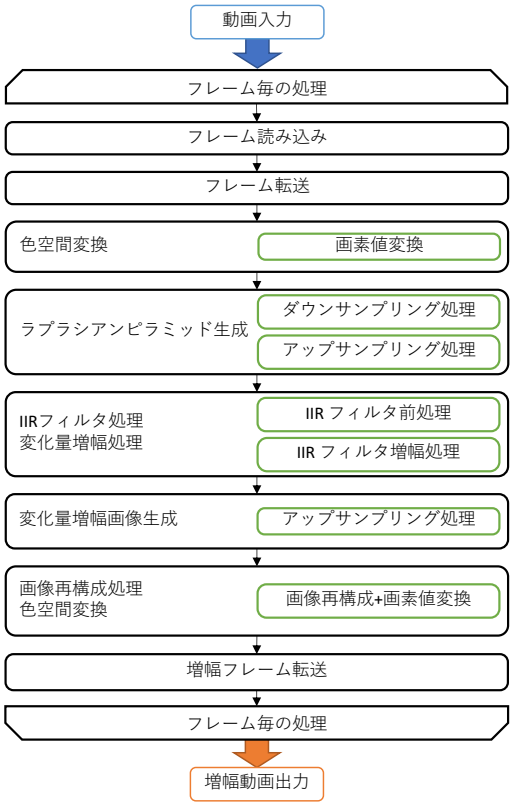


図 3　RGB 色空間におけるオイラー動画像誇張処理の処理フロー

のスケジューリング記述を利用した．

また，画像フレーム加工の各処理工程には異なる機能間で共通する処理アルゴリズムが存在する．そこで，このような処理を担う回路は共通化するために 1 つの処理単位としてまとめ，レジスタにフラグを設定することで処理を切り替えることとした．

オイラー動画像誇張処理は CIE-Lab 色空間に変換した上で増幅加工を行うが，これには浮動小数点の累乗計算が必要である．これは回路資源を著しく消費する上，オイラー動画像誇張処理の構成に本質的に必要な処理ではない[*1][*2]ため，本稿における実装では RGB 色空間で処理を行うオイラー動画像誇張処理のハードウェア実装を行う．

図 2 で取り上げた各処理工程をハードウェア化する機能毎に分割した場合のオイラー動画像誇張処理の処理フローを図 3 に示す．図 3 の緑で示されるのがハードウェア化する機能であり全 6 種類の IP の設計を行う．この時，画像フレーム 1 枚に対してアップサンプリング処理はラプラシアンピラミッド生成の一部と変化量増幅画像生成で共通利用できる IP として設計する．次フレーム読み込みやハードウェア制御，メモリ管理はソフトウェアによって処理する．

また，本研究では入力は動画ファイルからデータを読み込み，動画ファイルを出力する実装を行う．カメラ入力を

[*1] CIE-Lab 色空間を用いたほうがより自然な処理結果となる．
[*2] 実行時間に与える影響は大きいため，色空間変換処理を含めて加速実行したほうがフレームレートの改善率は大きくなると予想される．

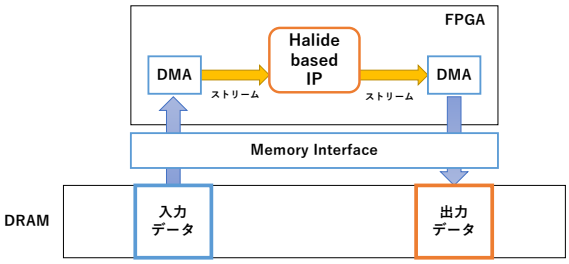


図 4　FPGA を用いた高速実行

利用可能な実装は今後の課題とする．

4.2　FPGA による加速実行

本研究では，画像データに対して処理を行うために FPGA を用いたストリーム処理を行うことで高速化を図る．ある出力値の計算に複数の座標のデータが必要な場合などは，Halide コンパイラ FPGA バックエンドがそれらの相対的な座標を計算し，十分なサイズのラインバッファを IP 内に生成する．

図 4 に本研究における FPGA による加速実行の基本的な構成を示す．FPGA に実装された IP へのデータ転送は DMA を用いる．主記憶から DMA によりデータを読み出し，順次データを IP に転送し処理を行う．IP で処理されたデータは DMA によってメモリに書き込まれ，全データが転送されると，処理が終了する．

本稿の設計では，6 種類の IP にそれぞれ入出力用の DMA エンジンを用意し，IP 間の入出力は主記憶を介して行う．すなわち，入力データを生成する IP（先行する処理を担う IP）の出力が書き込まれた領域を後続の処理を担う IP が読み出す．そのため，後続の処理を担う IP の起動は先行する処理を担う IP の最後のデータに対する処理が完了した後とする．

4.3　ハードウェア制御

本研究では，ユーザ空間メモリ領域から DMA 可能なメモリ領域にアクセスする仕組みとして udmabuf (user space mappable DMA Buffer) [6] カーネルモジュールを利用する．これにより，DMA 可能なメモリ領域の確保とその領域へのユーザ空間からのアクセスを実現する．また，ユーザ空間メモリ領域から Halide を元に生成した IP および DMA エンジンの制御を行うために UIO (Universal I/O) を用いた．この時 UIO からは IP の制御に加えて DMA の入出力データサイズを制御する．

4.4　Halide による実装

4.4.1　画素値変換

ここでは RGB 画像フレームの各画素値が持つ 0 以上から 255 以下の整数値データを 0 以上 1 以下の float 型のデー

$$\frac{1}{255}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

図 5 ガウシアンフィルタのカーネル

タに変換するために 255 で各画素値を割ることで実現できる．

```
dst(c, x, y) = cast<float>(src(c, x, y))/255.0f;
```

4.4.2 ダウンサンプリング処理

ダウンサンプリング処理ではじめに図 5 に示すカーネルを用いたガウシアンフィルタによるコンボリューションを行う．

```
Func clamp;
clamp = BoundaryConditions::mirror_interior(src,
    0, src_c, 0, src_w, 0, src_h);
RDom r(-2, 5, -2, 5);
Func gau;
Expr c1 = select(abs(r.x) == 2, 1, abs(r.x) == 1,
     4, 6);
Expr c2 = select(abs(r.y) == 2, 1, abs(r.y) == 1,
     4, 6);
gau(c, x, y) = Halide::Element::sum_unroll(r, c1
    * c2 * clamp(c, x + r.x, y + r.y));
Func output;
output(c, x, y) = gau(c, 2 * x, 2 * y) / 256;
```

この時，Halide 特有の記述である RDom によるループ変数を用いる．sum_unroll 関数を利用して RDom で指定した次元でループ展開を行い，ループで計算される総和を Halide 特有の型である Func 型に格納を行う．このコンボリューション計算では，入力された画像領域外の画素値を利用し計算を行うため，領域外の画素値を BoundaryConditions を利用してデータ参照を可能にしている．本稿のダウンサンプリング処理では mirror_interior による 画像端折り返した参照を設定した．通常，Halide によるメモリ上のデータ配置は複数チャネル数を持つ画像データの場合，特定チャネルのデータ値が連続してメモリに配置された後，別チャネルのデータ値を並べるような配置となる．しかしながら，ダウンサンプリング処理ではコンボリューション時に注目画素位置周辺の各チャネルの画素値を計算に利用するため，メモリ上に並ぶ，ある画素位置の各チャネルの画素値のメモリ上の間隔は，入力画像の解像度に依存するため，入力画像の解像度をあげると画素値読み出し時のメモリアクセスを増大させてしまう．そこで，入力画像の特定画素位置の各チャネルの画素値はメモリ上に連続して配置することでメモリアクセス時間を大幅に短縮することができる．そのため Func 型に格納した画像データは全てチャネル成分からメモリ上に配置する．

4.4.3 アップサンプリング処理

アップサンプリング処理は入力解像度の異なる 2 つの画像データと 1 つのフラグを入力とし，1 つの画像データを出力する処理を構成する．入力解像度の小さい画像を画像 A ，大きい画像を画像 B とする．はじめに画像 A を画像 B と同解像度に拡大し，ガウシアンフィルタによりコンボリューションを行う．この時，画像境界部のコンボリューション計算で適切なデータ参照を行うために画像 A を横方向処理と縦方向処理に分割して計算を行う．

```
Func imgA_clamped1;
imgAx_clamped1 = BoundaryConditions::
    mirror_interior(imgA);
Func imgA_clamped2;
imgAx_clamped2 = BoundaryConditions::mirror_image
    (imgA);

Func mid2;
mid2(c, x, y) = pyrup_multi1<float>(
    imgAx_clamped1, imgAx_clamped2)(c, x, y);

Func imgAy_clamped1;
imgAy_clamped1 = BoundaryConditions::
    mirror_interior(mid2, 0, imgA_c, 0, imgA_w *
     2, 0, imgA_h);
Func imgAy_clamped2;
imgAy_clamped2 = BoundaryConditions::mirror_image
    (mid2, 0, imgA_c, 0, imgA_w * 2, 0, imgA_h);

Func mid3{"mid3"};
mid3(c, x, y) = pyrup_multi2<float>(
    imgAy_clamped1, imgAy_clamped2)(c, x, y);
```

次に，画像 A の処理結果と画像 B を用いて，入力されたフラグに応じた適切な出力結果を得るために Halide の関数 select による条件文を利用した．

```
Expr diff = cast<float>(imgB(c, x, y) - mid3(c, x
    , y));
Expr reco = cast<float>(mid3(c, x, y) + imgB(c, x
    , y));
Func output;
output(c, x, y) = select(flag == 0, diff, reco);
```

これにより条件分が真の場合と偽の場合で得られる出力を選択することができる．

4.4.4 IIR フィルタ前処理

IIR フィルタ前処理ではユーザは必要情報の増幅を行うために適切なカットオフ周波数を入力する必要がある．この時，高周波数帯と低周波数帯の処理結果をそれぞれ別に得る必要がある．

```
Expr temp = (1 - cutoff_freq) * lowpass(c, x, y)
    + cutoff_freq * src(c, x, y);
Func output;
output(c, x, y) = temp;
```

この処理結果は IIR フィルタ増幅処理で利用されるが，次フレーム処理時の IIR フィルタ前処理の入力としても利用される．

4.4.5 IIR フィルタ増幅処理

IIR フィルタ前処理における高周波数帯制限結果から低

表 1　評価に用いたハードウェア環境 (Avnet Ultra96V2)

CPU	Cortex-A53 64bit Quad-core processor 1.2 GHz
メモリ	2GB LPDDR4
LUT	70,560
FF	141,120
DSP	360
BRAM (36 Kb)	216

表 2　評価に用いたソフトウェア環境

OS	PYNQ Linux, based on Ubuntu 18.04 aarch64
C++ コンパイラ	g++ 7.3.0
Vivado / Vivado HLS	2018.2

表 3　ハードウェア資源使用量と使用率

	使用量	使用率
LUT	47,177	67%
FF	64,275	46%
DSP	121	34%
BRAM	46.5	22%

周波数帯制限結果を引くことによって増幅対象を適切に抽出する．ユーザは増幅係数を入力して抽出された結果との積を計算し，微小変化を増幅することができる．

```
1  Func temp;
2  temp(c, x, y) = high_cut(c, x, y) - low_cut(c, x,
       y);
3
4  Func output;
5  output(c, x, y) = cast<float>(max(0, temp(c, x, y
       ) * factor));
```

との積を計算し微小変化を誇張した増幅画像を得る．

4.4.6　画像再構成＋画素値変換処理

IIR フィルタ増幅処理結果をによる解像度の異なる増幅画像をアップサンプリング処理によって 1 枚の画像に再構築された結果と，増幅加工処理の画素値変換処理結果を入力とする．ここでは増幅加工処理の入力フレームに増幅画像の重ね合わせを行うことで入力動画の微小変化を増幅した出力値を得ることができる．

5.　FPGA 加速実行の評価

5.1　評価の概要と評価環境

本稿では，オイラー動画像処理を構成する処理を Halide コンパイラ FPGA バックエンドを用いてハードウェア化し，ソフトウェア処理とハードウェア処理を組み合わせた協調設計によりアプリケーションを構築した．まず，ハードウェア合成時の資源利用率を評価した．続いて，加速実行適用前後の処理結果が同じであることを検証した．さらに，FPGA 加速実行の効果を評価するため，加速実行適用前の OpenCV を用いたソフトウェア実行と FPGA 加速実行適用時のフレーム毎の処理時間を計測しフレームレートを算出し，比較した．

評価に用いたハードウェア環境を表 1 に，ソフトウェア環境を表 2 に示す．フレームレート評価には横 640，縦 480 ピクセルの 30 fps の全 301 フレームの動画ファイルを入出力とする．

5.2　ハードウェア合成結果

Halide で記述された各機能を Halide コンパイラ FPGA バックエンドを用いて IP 化し，デザインしたブロック図を図 6 に示す．また，このブロックデザインを 300 MHz での動作をターゲットに合成した際のハードウェア資源使用量と使用率を表 3 に示す．LUT の使用率が高いものの，他回路資源使用率は半分以下の割合であるため，オイラー動画像誇張処理のハードウェア処理部割合を高めた実装が可能であると予測される．そのため，このデザインより高速な処理をするオイラー動画像誇張処理のハードウェア実装の余地がある．しかしながら，今後色空間変換処理やカメラ入力のための回路を追加する予定のため，このデザインを採用する．

5.3　出力値の検証

Halide による設計では，Halide 記述レベル機能検証，Halide コンパイラ FPGA バックエンドが出力する C++ コードでの機能検証，そして，ハードウェア合成後の単体検証が可能である．これらは基本的に共通のテストプログラムで行うことができる．Halide 実装時にはこれらの検証行うことで所望の機能が実現されていることを確認できる．しかしながら，複数の機能をアプリケーションとして結合したときの検証は別途行う必要がある．そこで，アプリケーションの出力値の検証を行った．図 7 に入力動画の時間変化イメージと，ソフトウェア実行時と FPGA 加速実行時の出力動画の時間変化イメージを示す．

ほぼ同じ増幅動画が出力されるが，計算順序の変更などが原因でソフトウェア実行時の結果と FPGA 加速実行時の結果は 0.29%程度異なる．

5.4　フレームレート評価

ソフトウェア実行および FPGA 加速実行適用時のフレームレートはそれぞれ 7.84, 15.41 fps であった．FPGA 加速実行の適用により，ソフトウェア実行に比べフレームレートは 1.97 倍となった．

また，図 8 にフレーム毎の処理時間内訳を示す．FPGA 加速実行を適用した処理ついてはソフトウェア処理が 106.91 ms, FPGA 加速実行が 41.3 ms であり，2.59 倍の性能向上が得られている．ハードウェア–ソフトウェア間のデータ受け渡し操作である増幅フレーム転送は 12.92 ms と他の処理時間と比べて大きく，処理効率低下を招く一因となっている．本稿の実装では逐次的な処理による実装を行ったが，パイプライン実行によりフレーム間の並列性を活用し，

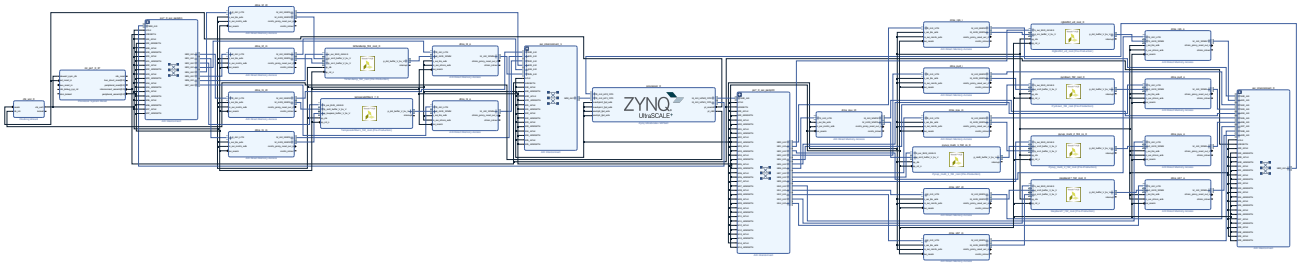
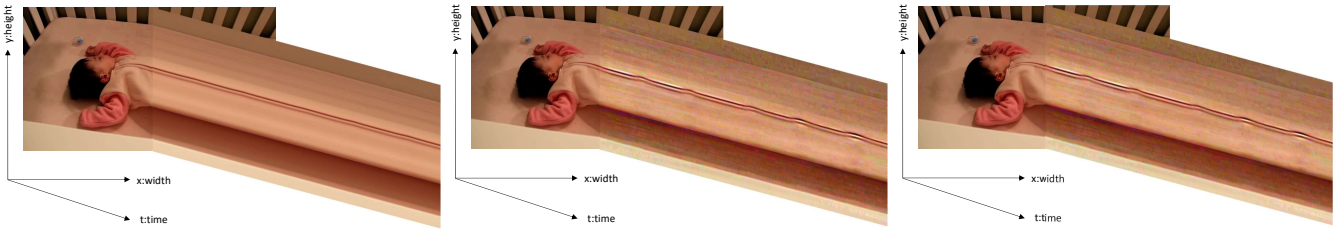


図 6　合成したハードウェアのブロック図



[1] 入力動画　[2] ソフトウェア実行　[3] FPGA 加速実行

図 7　入出力動画の時間変化イメージ

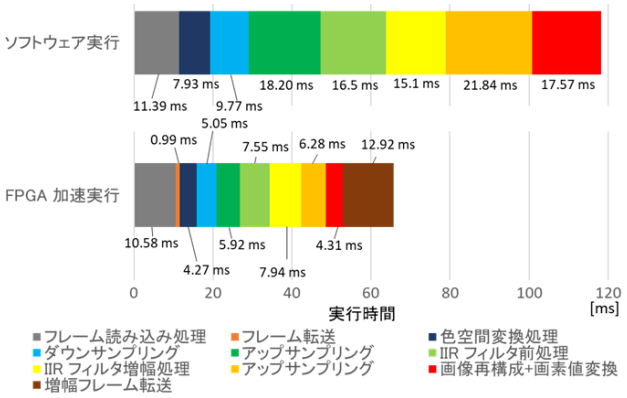


図 8　フレーム毎の処理時間内訳

処理時間を隠蔽して実装を行うことで処理時間を短くできる可能性がある．

6. おわりに

本稿では，オイラー動画像誇張処理を対象に Halide コンパイラ FPGA バックエンドを用いた FPGA 実装によるハードウェア処理と汎用プロセッサによるソフトウェア処理を組み合わせた協調設計による性能評価を行った．その結果，Halide コンパイラ FPGA バックエンドによる FPGA 実装と汎用プロセッサを組み合わせたオイラー動画像誇張処理のフレームレートは，オイラー動画像誇張処理の OpenCV 実装によるソフトウェア処理時のフレームレートより 1.97 倍の性能となることがわかった．しかしながら，目標としていた 30 fps を達成することはできなかった．

今後，本稿で実装を行わなかった CIE-Lab 色空間におけるオイラー動画像誇張処理，および，カメラから直接入力された動画に対してソフトウェア制御によるパイプライン処理を適用し高フレームレートによる動画出力を可能にする実装を目指す．また，GPU を用いた高速化の実装・評価についても今後の課題とする．

謝辞　本研究は一部，内閣府が進める「戦略的イノベーション創造プログラム（SIP）第 2 期／フィジカル空間デジタルデータ処理基盤」（管理法人：NEDO）による．

参考文献

[1] Andelson, E.H., C.H.Anderson, J.R.Bergen, P.J.Burt and J.M.ogden.: Pyramid Methods in Image Processing, *RCA Engineer*, Vol. 29, No. 6, pp. 33–41 (1984).
[2] Burt, P. J. and Adelson, E. H.: The Laplacian Pyramid as a Compact Image Code, *IEEE TRANSACTIONS ON COMMUNICATIONS*, Vol. 31, pp. 532–540 (1983).
[3] E.Poppel: *Grenzen des Bewustseins: Uber Wirklichkeit und Welterfahrung*, Verlages-Anstalt GmbH (1985).
[4] Hunter, R. S.: Photoelectric Color Difference Meter∗, *J. Opt. Soc. Am.*, Vol. 48, No. 12, pp. 985–995 (1958).
[5] Ishikawa, A., Fukushima, N., Maruoka, A. and Iizuka, T.: Halide and GENESIS for Generating Domain-Specific Architecture of Guided Image Filtering, *Proceedings of the 2019 IEEE International Symposium on Circuits and Systems*, ISCAS '19, pp. 1–5 (2019).
[6] Kawazome, I.: *u-dma-buf (User space mappable DMA Buffer)* (2017).
[7] Poh, M.-Z., McDuff, D. J. and Picard, R. W.: Non-contact, automated cardiac pulse measurements using video imaging and blind source separation., *Opt. Express*, Vol. 18, No. 10, pp. 10762–10774.
[8] Rosenfeld, A.: Some Uses of Pyramids in Image Processing and Segmentation, *Proceedings of the DARPA Imaging Understanding Workshop*, RCA Engineer, pp. 112–120 (1980).
[9] Vahid, F.: What is Hardware/Software Partitioning?, *SIGDA Newsl.*, Vol. 39, No. 6, pp. 1–1 (2009).

[10] Wu, H.-Y., Rubinstein, M., Shih, E., Guttag, J., Durand, F. and Freeman, W. T.: Eulerian Video Magnification for Revealing Subtle Changes in the World, *ACM Transactions on Graphics (Proc. SIGGRAPH 2012)*, Vol. 31, No. 4, pp. 13–15 (2012).
[11] Xilinx, Inc.: *Xilinx: UG902 - Vivado Design Suite User Guide: High - Level Synthesis (ver2018.2).*