Vol.2019-HPC-172 No.3
2019/12/18

Raspberry Pi上のVideoCore IVによる超解像処理Total Variation 正則化分離の実装と評価

竹田 大将[1]　近藤 鯛貴[1]　佐藤 裕幸[1]　杉野 栄二[1]

概要：我々は低コスト・省電力・省スペース化を目指し Raspberry Pi 上の GPGPU により，高負荷計算実現が可能か研究を行っている．今回はその Raspberry Pi GPGPU の 1 応用として，超解像処理システムを構築することを検討した．超解像技術は様々な手法が提案されているが，中でも Total Variation (TV) 正則化分離を用いた手法は最も有望なもののひとつと考えられる． TV 正則化分離は，入力画像を低周波成分と エッジ成分から構成される骨格成分，高周波成分とノイズから構成されるテクスチャ成分に分離する処理である．本稿では安価でありながら理論性能 24GFLOPS の GPU(VideoCore IV) が搭載されている特徴を持つ Raspberry Pi にて超解像処理の主要計算部のひとつである TV 正則化分離の GPGPU 化について評価を行った．その結果，CPU のみで演算を行う実装に比べて約 12 倍の高速化に成功し，汎用的な PC と比較しても Raspberry Pi GPU の価格性能比は圧倒的に高いことが確認できた．

キーワード：超解像処理，Total Variation 正則化，GPGPU

Implementation and Evaluation of Super Resolution Total Variation Regularization Decomposition by VideoCore IV on Raspberry Pi

Abstract: We are studying whether high performance computing can be realized by GPGPU on Raspberry Pi aiming at low cost, power saving and space saving. As one application of the Raspberry Pi GPGPU, we examined the construction of a super-resolution processing system. Various methods have been proposed for super-resolution technology, but the method using total variation (TV) regularization separation is considered to be one of the most promising. TV regularization separation is a process that separates the input image into a skeleton component composed of low-frequency components and edge components, and a texture component composed of high-frequency components and noise. In this paper, we evaluated GPGPU for TV regularization separation, which is one of the main calculation parts of super-resolution processing, on Raspberry Pi equipped with GPU (VideoCore IV) with theoretical performance of 24GFLOPS. As a result, we succeeded in speeding up by about 12 times compared to the implementation that performs computation only with the CPU, and it was confirmed that the price / performance ratio of Raspberry Pi GPU was overwhelmingly high compared to general-purpose PCs.

Keywords: Super resolution, Total Variation Regularization, GPGPU

1. はじめに

近年，NVIDIA の Jetson Nano を中心とした GPGPU により，より低コストな高負荷計算の実現が盛んに行われている．我々はさらなる低コスト・省電力・省スペース化を目指し Raspberry Pi 上の GPGPU により，高負荷計算実現が可能か研究を行っている．Raspberry Pi は，ARM CPU と VideoCore GPU 内臓の SoC(System-on-a-Chip) を搭載したシングルボードコンピューターで，教育用途を中心に広く使われており，価格も安いもので 5 ドル程度で購入可能な安価なものである．Raspberry Pi 3 Model B+ までのモデルには理論性能 24GFLOPS の GPU が搭載されているが，NVIDIA CUDA のような開発環境の整備が進んでいないこともあり，GPGPU への適用例はまだ少ない．

[1] 岩手県立大学
Iwate Prefectural University

今回はその Raspberry Pi GPGPU の 1 応用として，超解像処理システムを構築することを検討した．画像の解像度を補間し，低下した成分を復元する超解像技術は，これまで様々な研究が行われてきた．中でも，Total Variation (以下 TV) 正則化分離法を用いて画像を成分ごとに分離し各成分特徴にあったフィルタを適用する手法は，超解像拡大に大きな効果があることが報告されている [1][2][3][4]．しかしながら，上記の手法は計算量が膨大でありコンピュータで解くには高いマシンスペックが要求されるため，実用するには高いコストがかかるという問題点がある．実際，近年の超解像技術を用いているとされる映像機器の多くは搭載プロセッサの性能・コストの観点から単純な線形補間による処理が多く，高周波成分の欠落により精細感は損なわれている．よって， 搭載プロセッサのコストを抑えつつ，画質劣化が発生しない TV 正則化分離法を用いた超解像処理を実現することが望まれる．本稿では，TV 正則化分離法を用いた超解像処理において，最も中心となる TV 正則化分離処理を Raspberry Pi Zero の GPU に実装し，その性能を評価したので報告する．本論文の構成は以下の通りである．まず 2 章では Raspberry Pi の概要．3 章において TV 正則化分離の概要について紹介する．4 章では Raspberry Pi 上の SoC 内蔵 GPU である VideoCore IV のアーキテクチャについて記載し，5 章にて VideoCore IV 上での TV 正則化分離の実装について述べ，6 章で提案法のパフォーマンス評価を示す．最後に 7 章をむすびとする．

2. Raspberry Pi

Raspberry Pi は Broadcom 社の SoC を搭載した小型のシングルボードコンピュータである．シングルボードコンピュータとしては圧倒的なシェアを持ち，教育用から産業用まで幅広く活用され，その性能は著しい発展を遂げている．SoC 内には ARM CPU と VideoCore シリーズ GPU，デバイスコントローラなどが内蔵されている．Raspberry Pi 各モデルと搭載チップを**表 1** に示す．

本稿ではこの中から最も安価で小型である Raspberry Pi Zero を実装環境に用いた．Raspberry Pi Zero は CPU プロセッサに 1GHz ARM1176JZF-S，GPU プロセッサに Broadcom VideoCore IV, 512MB の CPU/GPU 共用 RAM を搭載している．共用 RAM のパーティションは設定によって自由に割り当てる事ができる．CPU はシングルコアで 1GHz 駆動とそこまで性能が高いとは言えない

表 1 Raspberry Pi 各モデルと搭載チップ
Table 1 Raspberry Pi models and chips.

	Raspberry Pi 1 系 Zero 系	Raspberry Pi 2 Model B	Raspberry Pi 3 系	Raspberry Pi 4 系
チップ	BCM2835	BCM2836 BCM2837	BCM2837 BCM2837B0	BCM2711
GPU	VideoCore IV	〃	〃	VideoCore VI
CPU	ARM 1176JZF-S	ARM Cortex-A7 ARM Cortex-A53	ARM Cortex-A53	ARM Cortex-A72

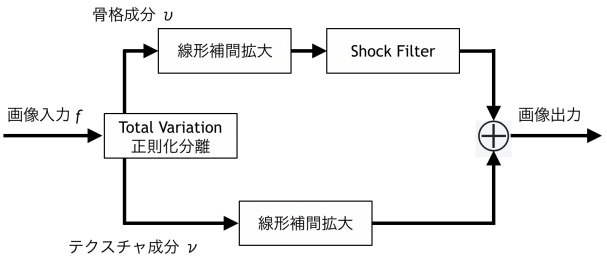

図 1 超解像の処理フロー

が，GPU プロセッサに VideoCore IV を搭載しているのが大きな特徴であり，これを活用することによって並列計算においては計算速度の向上を図ることができる．低コスト・省電力・省スペースなシステムとして，最適なデバイスである．

3. TV 正則化分離の概要

TV 正則化分離法を用いた超解像処理の概要を図 1 に示す [5]．この中で，最も中心となる機能が TV 正則化分離処理である．TV 正則化分離は，入力画像を低周波成分とエッジ成分から構成される骨格成分，高周波成分とノイズから構成されるテクスチャ成分に分離する処理である．

TV 正則化分離には Rudin，Osher，Fatami らによって提案された ROF モデル [3] と呼ばれるものが用いられる．ROF モデルは，式 (1) に示される評価関数 $F(u)$ を最小化するものである．

$$F(u) = TV_u + \lambda \sum_{i,j}^{P,Q} (u_{i,j} - f_{i,j})^2 \qquad (1)$$

ここで，$f_{i,j}$ は入力画素値，$u_{i,j}$ は演算出力画素値，P, Q は入力画像の縦と横の画素数，λ は正の定数である．式 (1) の右辺第 1 項は TV 項，第 2 項は拘束項と呼ばれている．TV 項は式 (2) のように隣接画像の差分ノルムで与えられる．

$$\begin{aligned} TV_u &= \sum_{i,j}^{P,Q} |\nabla_u|_{i,j} \\ &= \sum_{i,j}^{P,Q} \sqrt{(u_{i+1,j} - u_{i,j})^2 + (u_{i,j+1} - u_{i,j})^2} \end{aligned} \qquad (2)$$

ここで，勾配 ∇_u は式 (3) で与えられるベクトルを表す．

$$(\nabla_u)_{i,j} = ((\nabla_u)^1_{i,j}, (\nabla_u)^2_{i,j}) \qquad (3)$$

$$(\nabla_u)^1_{i,j} = \begin{cases} u_{i+1,j} - u_{i,j} & (i < N) \\ 0 & (i = N) \end{cases}$$

$$(\nabla_u)^2_{i,j} = \begin{cases} u_{i,j} - u_{i,j+1} & (j < N) \\ 0 & (j = N) \end{cases}$$

評価関数 $F(u)$ を最小化する $u_{i,j}$ を求めるとき，一般には

(a) 元画像

(b) 骨格成分

(c) テクスチャ成分

図 2 TV 正則化分離の適用

収束の速い Chambolle の射影法 [6] が用いられる．Chambolle の射影法は式 (4) のように射影法を用いた反復計算により p が十分に収束した時，式 (5) により骨格成分 $u_{i,j}$ とテクスチャ成分 $v_{i,j}$ を得ることができる．

$$p_{i,j}^{(t+1)} = \frac{p_{i,j}^{(t)} + \tau\{\nabla(\mathrm{div}p_{i,j}^{(t)} - f_{i,j}/\lambda)\}}{1 + \tau|\nabla(\mathrm{div}p_{i,j}^{(t)} - f_{i,j}/\lambda)|} \tag{4}$$

$$v = \mathrm{div}p \quad u = f - v \tag{5}$$

ここで，p は初期値

$$p^{(0)} = [0,0]^T$$

の双対ベクトルである．また τ は最急降下法のステップサイズである．div$p$ は式 (6) のような範囲で求められる．

$$(\mathrm{div}p)_{i,j} = \tag{6}$$
$$\begin{cases} p_{i,j}^1 - p_{i-1,j}^1 & (1<i<N) \\ p_{i,j}^1 & (i=1) \\ -p_{i-1,j}^1 & (i=N) \end{cases} + \begin{cases} p_{i,j}^2 - p_{i,j-1}^2 & (1<j<N) \\ p_{i,j}^2 & (j=1) \\ -p_{i,j-1}^2 & (j=N) \end{cases}$$

Mandrill 画像を入力画像として，TV 正則化分離を適用した例を図 2 に示す．テクスチャ成分画像は見やすくするためにコントラストを強調している．

4. Broadcom VideoCore IV GPU の概要

VideoCore IV は Raspberry Pi の殆どのモデルに搭載されているモバイル向け GPU である．

VideoCore IV アーキテクチャは画像処理用の様々なユニットで構成されている [11] が，本節では GPGPU 用途で使用する場合に必要なユニットのみをピックアップして説明する．GPGPU 用途で使用する場合の主要なユニット構成を図 3 に示す．

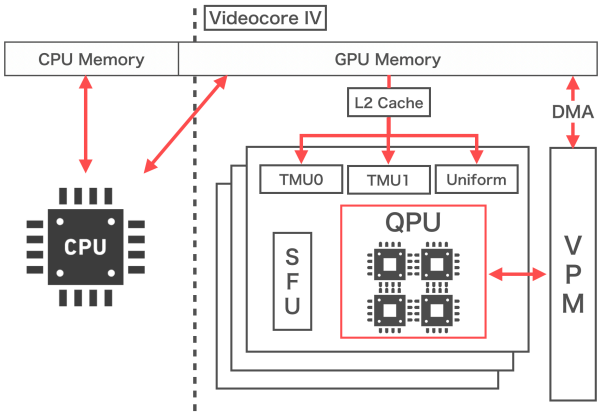

図 3 VideoCore IV の主要なユニット構成

演算コアである Quad Processing Unit (以下 QPU) コアはスライスと呼ばれるグループに編成されている．各スライスには 4 つの QPU コアが搭載されており，同じスライス内にある Texture and Memory lookup Units(以下 TMU), Uniforms Cache(以下 uniform), Special Functions Unit(以下 SFU) のリソースを共有する．VideoCore IV には合計 3 スライスが搭載されているので QPU コアは全体で 12 コアである．各 QPU コアは 32bit 16-way の SIMD プロセッサであり，内部的には 4-way の SIMD を 1 命令で 4 サイクル実行することで実現している．また，QPU コアは 1 命令で加算系と乗算系のベクトル演算を同時に実行できる dual-issue プロセッサである．以上のことから，GPU 全体で最大 $12\,(QPU) \times 4\,(cycle) \times 2\,(dual\,issue) = 96$ となり，1 命令で 96 の 32bit データ演算を行う事ができる．よって，VideoCore IV の動作周波数が 250MHz である Raspberry Pi Zero での理論性能は $96 \times 0.25 = 24 GFLOPS$ となる．

4.1 レジスタと特徴的な命令

各 QPU は 2 つのレジスタファイル，6 つのアキュムレータ，その他の 3 種類のレジスタを持つ．レジスタファイルは $32bit \times$ サイズ 16 の汎用的なレジスタをそれぞれ 32 個，計 64 個持っている．アキュムレータは $r0 \sim 3$ の 4 個が汎用的なレジスタ，$r4 \sim 5$ が読み取り専用の特殊レジスタとなる．レジスタファイルよりもアキュムレータのほうが高速であるが数が少ないため，アルゴリズム実装時には使用レジスタの節約を心がけ効率的に利用する必要がある．その他のレジスタは外部ユニットとの I/O を発行する I/O レジスタや，ハードウェアの情報の取得や設定を行う特殊レジスタが存在する．

また，特徴的な機能として SIMD の 0 番目の要素を複製する機能 (本稿では broadcast と呼ぶ) と，Horizontal Vector Rotation(以下 rotation) がある．rotation は $r0 \sim 3$ に格納されるベクトルに対して要素毎に巡回させる機能である．rotation で任意の数だけベクトルを巡回させた結果を

broadcast に書き込む事で，16 要素中の特定の 1 要素で埋められたベクトルを 1 命令で生成することが可能となる．元のレジスタを破壊することなく少ない命令数でスカラ値を扱えるため，この手法はプログラム内で頻繁に使用される．

4.2 特徴的なユニット

各スライスには特殊な計算機能をまとめた Special Functions Unit(以下 SFU) が搭載されている．I/O レジスタに値を書き込むことで，逆数，二進対数，底が 2 の指数関数，逆数平方根が計算可能．計算結果は r4 レジスタに格納される．しかしながら，r4 レジスタは後述する他のユニットにも頻繁に使用されること，演算によっては計算精度が低いこと，実行時間も遅いなどの理由から，なるべく使用するべきではない．また，1 つのミューテックス (mutex) と 16 個の 4bit セマフォ (semaphore) を持っているため，これによって QPU コアの同期制御・排他制御が実現できる．

4.3 データ転送

4.3.1 データ書き出し

メインメモリ・QPU 間のデータ転送には，L2Cash を介した 2 つの Texture and Memory lookup Units(以下 TMU)，uniform．加えて，メインメモリ・QPU 間のバッファ領域として Vertex Pipe Memory(VPM)，VPM とメインメモリ間でデータを転送する Direct Memory Access(DMA) の機能を使用する．まず，QPU からメインメモリへのデータの書き出しは，VPM を介した DMA によって行われる．TMU，uniform は読み出し専用のユニットであるため，データの書き出しはこの経路のみである．VPM は 64 × 16 のサイズを持った本来シェーダの頂点データを保持するバッファだが，GPGPU 用途では QPU で扱うデータを保持する汎用的なメモリ領域として使用する．VPM は VideoCore IV 全体で一つしかないため，各 QPU のアクセスを排他的に制御する必要がある．具体的にはデータの競合を避けるため，DMA によるデータ転送完了をポーリングするか，単純に処理を停止して待つなどアクセスを排他的に制御する．結果的に複数の QPU がアクセスする場合ボトルネックになってしまう問題がある．

4.3.2 データ読み出し

次に，メインメモリから QPU へのデータの読み出しについて，こちらも同じく VPM と DMA によって可能であるが，uniform と TMU を活用する事によってより高速に実現できる．uniform は，指定されたメインメモリのアドレスから 32bit のスカラ値を連続的に読み出すユニットである．uniform に，読み込みを開始する先頭アドレスを設定してから uniform の I/O レジスタにアクセスすると，指定したアドレスが指す値が 16 要素のベクトルとしてレジスタに展開される．読み込み開始アドレスを再設定しない限り，I/O レジスタにアクセスされるたび 4byte ずつメインメモリのアドレスを加算しながら値を読み込む．読み込み開始アドレスはカーネルプログラム実行時にホストが初期値を設定することができる．TMU は，本来 3D 描画でのテクスチャ座標補完などに利用されるユニットであるが，GPGPU 用途では 16 個の 32bit データをメインメモリから非同期的に読み出すために使用する．分散した 16 個の要素をとってくることも可能だが，多くの場合連続した 16 個のアドレスを指定する．それぞれが近いアドレスを指定することでキャッシュヒット率が上がり，より高速に読み出すことができるためである．また，非同期的に読み出しが行えるため，パイプラインを意識した設計が行いやすい．

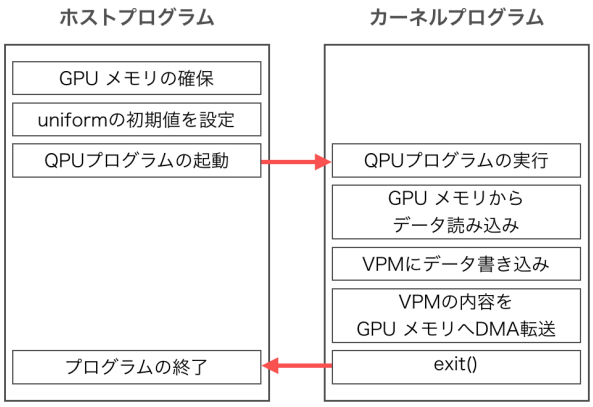

図 4 QPU プログラム実行の流れ

以上 2 つのユニットの特性を活かした高速なデータ読み出し手法について説明する．まず，カーネルプログラムに渡したい引数のアドレスをメモリの連続した領域に記録する．次に，引数が記録された領域の先頭アドレスをカーネルプログラム実行時に uniform に書き込む．こうすることで，uniform を読み出すたびに引数のアドレスが読み出され，そのアドレスを起点に TMU で 16 要素ずつデータを読み込む事で，DMA，VPM を使用する場合よりもより高速に連続領域の読み込みを行う事ができる．

4.4 QPU プログラム実行の流れ

基本的なプログラムの流れを図 4 に示す．

5. VideoCore IV での TV 正則化分離実装

5.1 実装の概要とスレッドの割当

TV 正則化分離を行う図 5 に示す処理を，GPU の特性に合わせて実装していく．本稿では 960x540(pixel) の qHD サイズ画像を入力画像と想定し，実装を行った．

図 6 ように 1 スレッドに 320x135 の領域を割り当て，計 12 スレッドで 1 つの画像を処理する．更に細かい単位で領域を分割しスレッドを割り当てる方法もあるが，VideoCore IV はスレッド間のデータ受け渡しは VPM かメインメモリを介して行うしかなく，図 5 の 4 行目のように処理画素

```
01 for(int iter=0; iter<t; iter++){
02     for(int i=0; i<HEIGHT; i++){
03         for(int j=0; j<WIDTH; j++){
04             int div=(p0[i][j]-p0[i-1][j])+(p1[i][j]-p1[i][j-1]);
05             Calc_img[i][j]=div-(F[i][j]/lambda);
06         }
07     }
08     for(int i=0; i<HEIGHT; i++){
09         for(int j=0; j<WIDTH; j++){
10             int nabla[2];
11             nabla[0]=Calc_img[i+1][j]-Calc_img[i][j];
12             nabla[1]=Calc_img[i][j+1]-Calc_img[i][j];
13             int norm=sqrt(pow(nabla[0],2)+pow(nabla[1],2));
14             p0[i][j]=(p0[i][j]+t*nabla[0])/(1+t*norm);
15             p1[i][j]=(p1[i][j]+t*nabla[1])/(1+t*norm);
16         }
17     }
18 }
19 for(int i=0; i<HEIGHT; i++){
20     for(int j=0; j<WIDTH; j++){
21         one=P0[i][j]-P0[i-1][j];
22         two=P1[i][j]-P1[i][j-1];
23         str[i][j]=src[i][j]-LAMBDA*(one+two);
24         tex[i][j]=src[i][j]-str[i][j];
25     }
26 }
```

図 5　TV 正則化分離処理の擬似コード

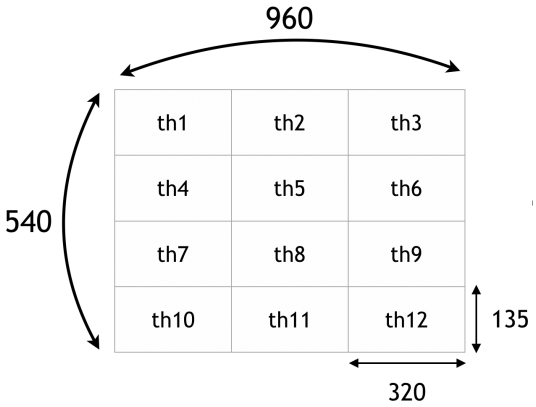

図 6　画像の分割

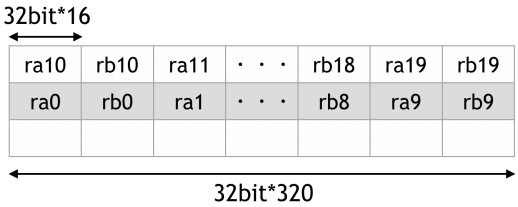

図 7　ブロック 1 行にレジスタ割付

の上下左右の画素を参照し計算する場面が多い TV 正則化分離では I/O によるボトルネックが大きくなり非効率である．よって，一定の幅，高さを持った領域をスレッドに割り当てることで，スレッド間のデータの受け渡しを減らしている．

5.2　上下の画素を参照する計算

各 QPU コアは図 7(色の付いている行が処理を行っている箇所) のようにレジスタファイル $ra0 \sim 9$，$rb0 \sim 9$ を使い 1 行処理するごとにメインメモリに転送される．

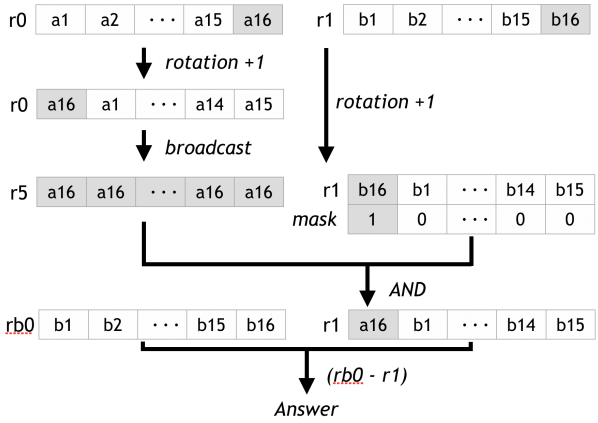

図 8　右に 1 要素シフトしたベクトルと比較する

また，次の処理行を読み込む前に $ra10 \sim 19$,$rb10 \sim 19$ に処理行の値を複製しておく事で，前の行の値として次のステップで参照する事ができるようにしている．これによって，式 (7) のような処理画素とその上下での演算を可能とする．

$$p0[i][j] - p0[i-1][j] \tag{7}$$

5.3　左右の画素を参照する計算

左右の画素を参照する計算には 4.1 でも解説した rotation と broadcast を使い特定の 1 要素で埋められたベクトルを 1 命令で生成する手法を応用している．

$$p1[i][j] - p1[i][j-1] \tag{8}$$

図 5 の 4 行目の第 2 項にある式 (8) を計算する流れを図 8 に示す．

このように，rotation によって左右にシフトしたベクトルを生成し，元のベクトルと演算することで，1 命令で 16 要素の演算を行う事ができる．mask をかけて AND を行うため，ベクトル a について rotation の結果を broadcast に書き込まず $r0$ に書き戻しても同等の演算が可能だが，broadcast された結果は $r5$ レジスタに書き込まれる特性から $r0$ レジスタを即座に開放することができるためレジスタの節約となる．一見工程が多く見えるが，rotation と broadcast は同時に実行できること，mask は事前に設定しておけることなどから，5 命令ほどで実現できる．

メインメモリへのデータの書き出しは，転送完了までの間に待ち時間がある．また，ラベルジャンプなど命令発行から実行完了まで数命令分のディレイがある命令がいくつかある．そこで，ソフトウェアパイプライニングとループ展開の組み合わせによって実行完了までの空き時間をなるべく隠蔽している．ソフトウェアパイプライニングは図 9 に示すように，ループを跨いで命令を並べ替え，ループを再構成することで，命令の並列性を高める最適化であ

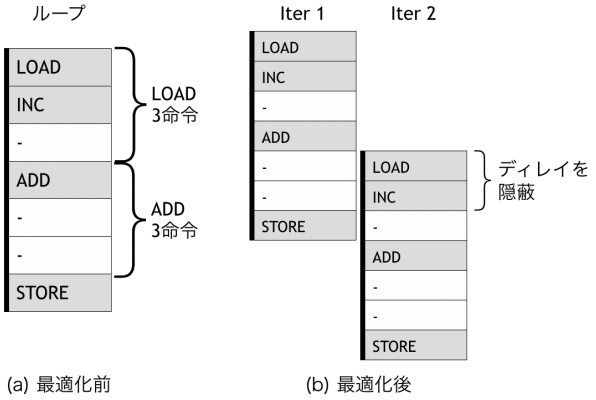

図 9 ソフトウェアパイプライニングの例

表 2 ソフトウェア環境

項目	ディストリビューション，バージョン
OS	Raspbian 9.4
Python	Ver.2.7.13
PyVideoCore	Ver.1.0.0
rpi-vcsm	Ver.2.0.0

る [7][8]．図 9 の (a) と (b) は，どちらも各命令の実行回数は同じになる．前処理と後処理を加えたループ展開を殆どのループ箇所で行っている．TV 正則化分離処理はループが多いため，この最適化手法は非常に効果的である．

VPM，DMA に関連する設定レジスタは VideoCore IV 全体で 1 つしか無いため，それらにアクセスする部分は semaphore による排他制御を行なってる．また，各 QPU の同期は mutex により実現している．

6. 評価実験

960x540(pixel) の画像に対して，Raspberry Pi Zero WH で TV 正則化分離処理を実行し，その実行時間を計測した．ソフトウェア環境は表 2 の通りである．

Raspberry Pi での GPGPU 開発環境は Idein 株式会社 [9] の rpi-vcsm と PyVideoCore[10] を使用した．rpi-vcsm は Raspberry Pi の VideoCore Shared Memory (VCSM) の Python ドライバである．PyVideoCore は Raspberry Pi ボード上の GPGPU 用の Python ライブラリである．Broadcom VideoCore IV 用の QPU アセンブラはいくつかあるが，PyVideoCore はアセンブリ言語が Python 言語の内部 DSL として実装されており，ホストプログラムと GPU カーネルプログラムを 1 つの Python スクリプトにまとめることができる．そのため，事前コンパイルなしでプログラムを実行できる，Python の機能，既存のライブラリ・ツール資産を利用することができるなどの点から，比較的簡単に開発できる．

図 10 を入力に，実際に Raspberry Pi Zero で TV 正則化分離を行って得られた骨格成分とテクスチャ成分をそれぞれ図 11，12 に示す．それぞれ成分が分離できている事が確認できる．

表 3 TV 正則化分離の実行時間比較

CPU(1th)[msec]	GPU(12th)[msec]	CPU/GPU[倍]
214.939	17.719	12.130

図 10 元画像

図 11 骨格成分

図 12 テクスチャ成分

また，Raspberry Pi Zero の CPU のみで TV 正則化分離を行った場合と，GPU を活用した場合の実行時間の比較を表 3 に示す．どちらも，収束に向けた計算反復回数は 1 回である．ここから，CPU 実装に比べ約 12 倍高速化できている事が確認できる．

汎用的な PC 上 (CPU は Intel Core i7-7700K[12]) で図 1 に示した超解像処理を全て実行し，実行時間を計測した．実行環境を表 4 に示す．また，そこから推測される Raspberry Pi の超解像処理実行時間を表 5 に示す．

どちらも，TV 正則化分離の計算反復回数は 1 回である．本提案の TV 正則化分離の実行時間は 17.719[$msec$]，Core i7-7700K での実行時間は 2.834[$msec$] であり，その比は約 6.25 倍である．よって，Core i7-7700K での実行時間をそれぞれ 6.25 倍することで，超解像全体を Raspberry

表 4　汎用 PC 環境

CPU	Intel Core i7-7700K
CPU コア数	4
CPU 周波数	4.20〜4.50 GHz
CPU ソケット数	1
メインメモリ	16GB
コンパイラ	g++ 7.4.0
コンパイルオプション	-O3 -fopenmp

表 5　超解像処理の実行時間推測 [msec]

560x960->1080x1920	TV 正則化分離	線形補間拡大	Shock Filter	ADD	Total
Raspi-GPU(12th)	17.719	30.606	54.169	4.806	107.300
i7-7700k(8th)	2.834	4.897	8.667	0.769	17.167

Pi の GPGPU で行なった場合の実行時間を推測した．その結果，超解像処理全体を Raspberry Pi の GPGPU で行なった場合，107.3[$msec$] かかると推測される．この処理時間は，30FPS の動画への適用は難しいが，毎秒画像を切り替えるようなディジタルサイネージには十分である．Raspberry Pi の CPU を使用するだけでは，約 1/12 の性能になるので，毎秒の画像切り替えには対応できない．

また，本提案は実行時間のみで比較すると Core i7-7700K に約 6.25 倍劣っている．Core i7-7700K を搭載した PC の価格は 約 13 万円程度であり，Raspberry Pi Zero の価格は 650 円程度なので，その比率は 200 倍であり，性能比の 6.25 倍より遥かに大きい．Core i7-7700K を搭載した PC をディジタルサイネージに用いるのはオーバースペックであるが，汎用 CPU の価格性能比はあまり変わらないと仮定すると，より低性能な汎用 CPU を使用した場合でも，Raspberry Pi GPGPU の価格性能比は圧倒的に高いと言える．

7. むすび

本稿では，TV 正則化分離法を用いた超解像処理において，最も中心となる TV 正則化分離処理を Raspberry Pi の GPU に実装することで，CPU のみで演算を行う実装に比べて約 12 倍の高速化に成功した．汎用的な PC 上 (CPU は Core i7-7700K) の実装と実行時間のみで比較すると約 6.25 倍劣っているが，価格比が 200 倍であることを考慮すると価格性能比は圧倒的に高いことが証明できた．よって，実用的には毎秒画像を切り替えるようなディジタルサイネージなどの安価な構築に適していると言える．今後の課題として，さらなる高速化と超解像処理フロー全体を GPGPU で実現することを検討している．また，今年 (2019) 一般に発売された Raspberry Pi 4 には VideoCore IV の次世代アーキテクチャである VideoCore VI GPU が搭載され，更なる実行速度の向上が見込まれるためこちらでの研究も求めたい．

参考文献

[1] 齊藤 隆弘: 1 枚の画像からのオーバーサンプリング，映像メディア学会誌 Vol. 62，No. 2，pp. 181-189 (2008).
[2] Tomio Goto，Takafumi Fukuoka，Fumiya Nagashima，Satoshi Hirano，Masaru Sakurai: *Super-Resolution System for 4K-HDTV*，International Conference on Pattern Recognition，DOI 10. 1109/ICPR，pp. 4453-445 (2014).
[3] LI. Rudin，S. Osher，E. Fatemi: *Nonlinear total variation based noise removal algorithms*，Physica D，vol. 60，pp. 259-268 (1992).
[4] 鶴崎 祐貴，亀田 昌志，プリマオキディッキアルディアンシャー：Total Variation 正則化を用いたテクスチャの鮮鋭化のための単一画像による超解像，画像電子学会誌 Vol. 46，No. 1，pp. 206-217 (2017).
[5] 渡辺 将史，長島 史弥，作田 泰隆，後藤 富朗，平野 智，桜井 優：ショックフィルタと TV 正則化フィルタを用いた拡大画像の高精細化，映像情報メディア学会技術報告，Vol. 37，No. 15，pp. 1-4 (2013).
[6] A. Chambolle: *An algorithm for total variation minimization and applications*，J. Mathematical Imaging and Vision，vol. 20，pp. 89-97 (2004).
[7] 中田 育男：ソフトウェア・パイプライニングの一実現法，情報処理学会論文誌，Vol. 47，No. SIG 16，pp. 44-51 (2006).
[8] Jason Robert Carey Patterson：*Basic Instruction Scheduling(and Software Pipelining)*，Lighterra (2001).
[9] Idein 株式会社：Idein Inc.，`https://idein.jp/`，(2019.11.11).
[10] 中村 晃一：Raspberry Pi で GPGPU，`https://qiita.com/9\_ties/items/2e85318989170f967e4b`，Qiita (2019.11.11).
[11] Broadcom: *VideoCore® IV 3D Architecture Reference Guide*，(2013.09.16).
[12] インテル ® Core™ i7-7700K プロセッサー，`https://ark.intel.com/content/www/jp/ja/ark/products/97129/intel-core-i7-7700k-processor-8m-cache-up-to-4-50-ghz.html`，Intel (2019.11.13).