6C-4

多段階画像処理におけるメモリアクセス削減の検討

高橋　幸恵†　　今田　敬†　　境　隆二†　　加藤　宣弘†

株式会社　東芝　コアテクノロジーセンター†

1．　はじめに

ハイビジョン等の画像処理をプロセッサで行う場合、高速にアクセス可能なローカルメモリのサイズには制限があるため、通常ブロック分割を行って演算する。しかし、画像処理を構成する処理が多段階にわたる場合、メインメモリへのアクセスがボトルネックとなって性能が劣化する。そのため、ブロック化した画像に対して、複数フェーズをローカルメモリ上で実行する手法が考えられるが、ブロック境界における演算が重複して発生し、演算量が増大するという問題が生じる。

本稿では、ブロック化したデータをローカルメモリに効率的に配置することで、演算量の増大を抑制しつつメインメモリへのアクセスを削減する方法について考察する。

2．　多段階画像処理

本稿では、下記の図 1 のように斜線で囲まれた部分のように複数回の画像処理を適用するものを多段階画像処理と呼ぶことにする。

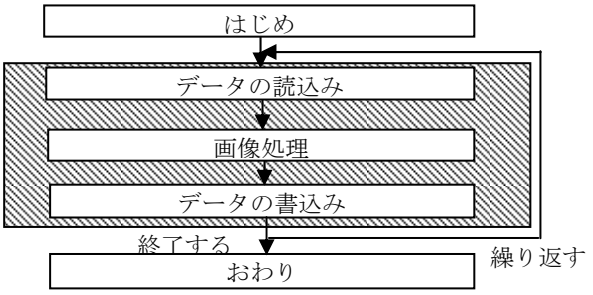

図 1）多段階画像処理のフロー

今回は、その一例である Total Variation による非線形画像分離のアルゴリズムを対象とし、Cell Broadband Engine™ (以下 Cell)上で評価した。

3．　Total Variation

Total Variation フィルタは、ノイズ除去アルゴリズムの 1 つで、ノイズの統計的情報を用いて修復画像の総変動(total variation)を最小化する手法である。またノイズ除去に限らず、正則化に基づく画像修復技術としても応用されており、画像の骨格成分(ノイズ除去画像)とテクスチャ成分(ノイズ画像)に分離する方法としても利用されている。今回は、骨格成分とテクスチャ成分の分離を目的としてプログラムを実装し評価した。

入力画像 I を対数変換し、入力画像対数関数 f として、関数 f を対数骨格関数 u と対数テクスチャ生成子関数 v との和に分解する。主な処理の流れは図 2 のようになる。

A study of memory access reduction in multistep image processing

†Yukie Takahashi, Kei Imada, Ryuji Sakai, Nobuhiro Kato ・Core Technology Center, TOSHIBA Corporation

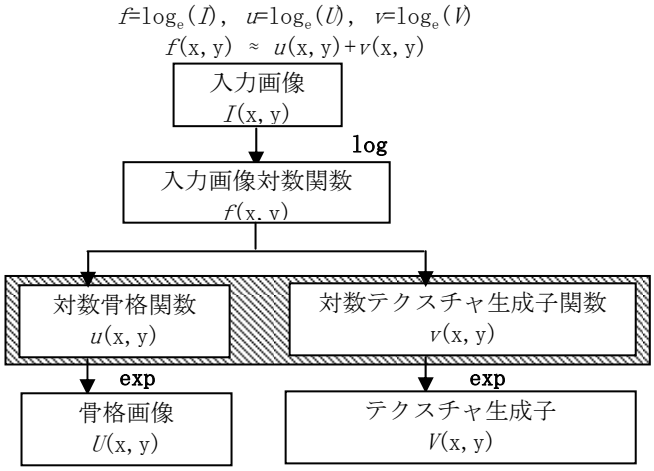

図 2）骨格成分とテクスチャ成分の分離フロー

この Total Variation では、斜線で囲まれた各関数処理の中で、周囲画素情報を用いて濃度変化を調べるフィルタ処理(図 3 参照)が多段階的に行われている。一般的には、値が既定値に収束するまでループされるが、今回は定数回(4 回)のループとして実装した。またこの処理は、Float 型で処理されることから、演算量・データ転送量共に負荷が高い。

4．　Cell/SPE の特徴

Cell は、1 個の PPE と 8 個の SPE からなるマルチコアプロセッサである。

SPE は主に、SPU と呼ぶ SIMD プロセッサ、SPU のコードと SPU が直接ロード/ストア可能なデータを保持する LS(Local Storage)、LS とメインメモリの間でデータを DMA 転送する MFC(Memory Flow Controler)から成る。SPU が直接参照可能な LS は、256KB とサイズに限りがあるため、メインメモリと LS の間で適宜データを DMA 転送する必要がある。

5．　メモリアクセス削減に関する設計

今回は 1 行分のデータを 1 ブロックとして設計した。各 1 ブロックを処理する場合、データの依存関係は図 3 のようになる。前ループの 3 行分(同一行と上下各 1 行)のデータを参照して 1 ブロック分の演算を行う。1 章で述べた通常のブロック分割を適用して、Loop_cnt 毎に、1 フレーム分の DMA 転送と演算を行うと、演算の重複は最小限に抑えられるが、メモリアクセスが多いという問題がある。本稿では、上記を従来手法とし、今回は以下 2 点について見直し、設計した。

(1)メモリアクセスを削減するために、メインメモリから読み込んだデータ・各ループで得られた演算結果を後ループで使い続ける。〈メモリアクセスの削減〉

(2)各ループの処理は、前ループの演算結果を用いていて、処理する対象行のデータのみならず上下行のデー

タも必要なため、各ループの処理対象ブロックの位置をずらして演算し途中結果は LS に保存する。<重複演算の抑制>

各データの相関関係は、図 4 のようになる。点線で囲まれた部分は、各ループの処理対象データを示している。対象データを求める際、上の行データが必要なため LS に保存されている前ブロックの処理結果を参照する(楕円で囲まれた部分)。下の行データも演算に必要なため、初回のデータのみ(Loop_cnt=0 の●)、メインメモリから DMA 転送し、そのデータを参照して新規に得られたデータを後段に流用し、最終ループのデータのみメインメモリ に書き戻す(Loop_cnt=4 の●)。

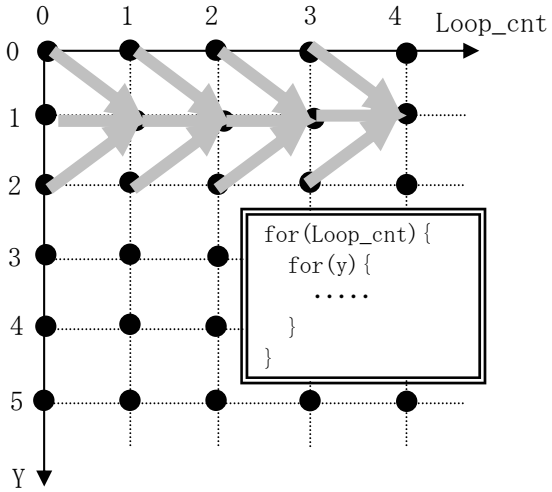

図 3) 処理対象データと参照データの関係 (*1)

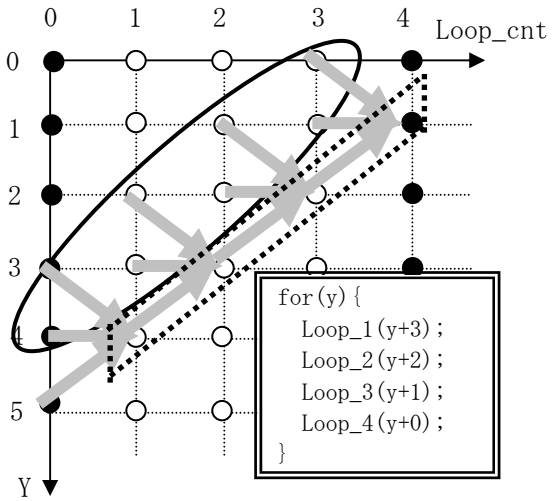

図 4) ブロック単位での処理の流れ(提案手法) (*1)

(*1) ○:DMA 転送されないデータ
●:DMA 転送されるデータ

6. 評価と考察

まず、実行時間の内訳を表 1 に示す。

	従来手法	提案手法
DMA 転送	7.676	0.532
演算	7.900	6.800
TOTAL	15.576	7.332

表 1) 実行時間の内訳 [ms/frame]

表 1 より、DMA 転送については約 14 倍の性能向上ができた。冒頭でも述べたように、多段階画像処理はメモリアクセスが多く、今回実装した Total Variation も、60fps で処理する場合のメモリバンド幅が、従来手法では約 5.7[GB/s]だったが、5 章で設計したように、データをローカルメモリに効率的に配置することで、0.3[GB/s]まで削減することができ、上記のような性能向上に繋がった。また、演算部は構成を変更することで、命令の発行時に生じていたストールが削減されることなどにより性能が向上した。

次に、アプリケーションが並列に動作することによって、メモリアクセスが増加した場合の多段階画像処理の実行時間を評価し、図 5 に示す。今回は、DMA 転送のみを行うダミープログラムをアプリケーションとして用意する。(並列実行するアプリケーションの数=0 の実行時間は、多段階画像処理を単体で動かした時の性能である)

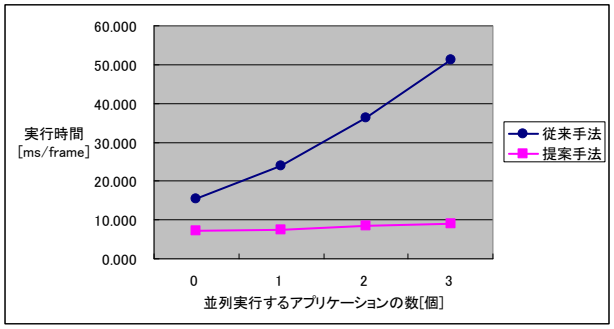

図 5) 並列実行するアプリケーションの数と多段階画像処理の実行時間の関係

図 5 より、今回提案した手法では、メモリアクセスが増加した場合の実行時間の増加が、従来手法と比較して小さいことが分かる。従来手法では、DMA の転送量が多く演算のバックグラウンドで必要なデータを転送しきれないため、並列に実行されるアプリケーションの数の増加によってメインメモリへのアクセスが増えた場合、DMA の転送時間が増大していた。しかし、今回提案した手法では DMA 転送を、初回と最終回のみで行うように変更することで、DMA の転送量自体を削減できたため、演算のバックグラウンドで必要なデータを転送でき全体の実行時間が大きく増大しなかったと思われる。

7. おわりに

本稿では、ブロック化したデータをローカルメモリに効率的に配置することで、演算量の増大を抑制しつつメインメモリへのアクセスを削減する方法について提案した。その結果、従来手法と比べてメインメモリへのアクセスの削減・演算の効率性の向上を図ることができた。

本手法は、Cell と同様にローカルメモリを持つ GPGPU や、キャッシュメモリをもつ汎用プロセッサにおいても有用である。

参考文献

[1]V. Caselles, A. Chambolle, M. Novaga, "The discontinuity set of solutions of the TV denoising problem and some extensions"
[2]http://cell.scei.co.jp/j_download.html
・CBE_Architecture_v10_j.pdf
・SPU_language_extensions_v21_j.pdf