4H-3

CUDAによるフラクタル画像符号化の実装

目出雅之†　田中宏二郎†　若谷彰良††
†甲南大学大学院　自然科学研究科
††甲南大学　知能情報学部　知能情報学科

1．はじめに

CPUの演算性能の向上は，数年前からスローダウンしつつあるのに対して，GPUはいまだに性能が向上し続けており,すでにCPUの汎用性能をはるかに超えている．

そこで，コンピュータの処理を高速に処理させる方法に，グラフィックボードに搭載されている GPU (Graphics Processing Unit) を用いた処理が注目されている．

フラクタル画像符号化は高い圧縮性能が得られるアルゴリズムであるだけでなく，画像検索のように他のアプリケーションにも応用されるものであるが [1],その実行時間は長く，実用化にとっては高速化が不可欠である．そこで,本論文では,高速化を目的として,CUDA (Computer Unified Device Architecture) を実装し,その評価をおこなう [2].

2．CUDA

今回はGPUに幅広く使用されているNVIDIA社のCUDAを用いて実験を行う．

特徴としてはSIMD (Single Instruction Multiple Data) 型演算であるという点であり,これは1つの命令で複数の膨大なデータに対して，同じ処理する演算手法である．低いクロック周波数でも性能が高い点が長所であるが，分岐が多いなど複雑な制御構造をもつ処理では，アイドルになるプロセッサが発生するという点が短所である [2].

3．フラクタル画像符号化

フラクタル画像符号化とは図形の一部分が他の一部分に似ていると仮定して圧縮する．また,反復によって極限画像に収束させることによって復号する．

フラクタル画像符号化は,膨大な量の符号化計算がある．また，符号化アルゴリズムはどのレンジブロックでも同じなのでSIMD型のCUDAは適している．しかし，レンジが固定されている場合は符号化するレンジブロックが連続しているので問題にならないが，最適な圧縮を行うにはレンジを適応的に変える(今回は3段階) 必要があり，符号化するレンジブロックが連続でなくなる可能性が高い．

そこで，提案手法ではあるレンジで符号化されなかったレンジブロックは新たに間接ベクトルを用意し，連続に並ぶように並び替えをし,CUDAで効率良く計算できるようにする．一般に，間接ベクトルを用いたメモリアクセスではコアレスアクセスを阻害することになるので，性能に悪影響を与えるが，今回のアプリケーションでは，そのアクセスは多重ループの最内側ループ内には存在しないので，性能に与える影響は少ないと考えられる．

4．実験

4.1 実装方法

具体的な圧縮の方法は,まず画像を16

Implementation of fractal image coding by CUDA
Masayuki Mede,Koujiro Tanaka,Akiyoshi Wakatani,
†School of Natural Science ,Konan University
††Faculty of Intelligence and Informatics,Konan University

×16 ずつに区切りこれをレンジ 16 と呼ぶ. また区切られた領域をレンジブロックとする. これとは別にレンジブロックの４つ分の正方領域（レンジ 16 では 32×32）をドメインブロックとする. すべてのレンジブロックに対してすべてのドメインブロックの画像に対する類似度を判定する. なお, ドメインブロックは 90° ごとの回転と反転の計８通りのパターンで比較する. 基準以上に似ているドメインブロックが存在する場合, そのレンジブロックの符号化を終了する. 符号化できなかった場合は, レンジブロックを四分木分割(Quadtree Partition)し, レンジを8,4と順次, 縮小して符号化する. レンジ４では残ったレンジブロックすべて 符号化する [3].

if 文を使った単純な符号化(方法１)では, SIMD 型の演算方法のために符号化したレンジブロックとしなかったレンジブロックを同じように並列処理させることになり, アイドルとなるコアが存在することになり並列処理の性能を最大限に使うことができない. そこで, 符号化しなかったレンジブロックが連続になるように並び替えを行う. 間接ベクトルを用いて, 符号化を行う(方法 2).この２つの方法を以下の環境で評価する.

4.2 実験内容

表１に示す実験環境において, 横 512×縦 512 の lenna 標準画像をフラクタル符号化し, その性能を評価する.

表１ 実験環境

OS	Windows 7
CPU	Atom330 (1.6GHz×2)
メモリ	2.00
GPU	ION(1.1GHz)
SPの数	2×16
CUDAのバージョン	3.0
開発環境	Visual Studio 2008

4.3 実験結果と考察

表2 レンジを適応的に変化させたときの性能比較

	符号化時間(s)	スピードアップ(倍)
CPUのみ	913.7	-
方法1	70.3	13.00
方法2	16.24	56.26

各レンジにおいて, CUDA を利用することで CPU のみでフラクタル画像符号化をするより 10 倍以上の性能の差がひらく. レンジ 16 が他のレンジに比べて性能差が小さいのはシェアードメモリが最大限に活用できずに occupancy が低いためだと考えられる.

表２に示すとおり, 方法 2 は間接ベクトルに並べ替えるコスト及び, 間接ベクトルを用いたメモリアクセスによるコスト増があるが, GPU のコアを有効利用できるので, 方法１よりも高い性能を得ることができた.

5. おわりに

間接ベクトルを用いた GPU の CUDA 実装により, Atom のような低スペックの PC でも適応的なレンジブロックのサイズを用いたフラクタル画像符号化で性能向上を実現できた.

参考文献

[1] 横山貫之, 渡辺俊典, 菅原研, 上野芳朗, “フラクタル符号に基づく構造的な類似性抽出手法の提案,” 信学技研, PRMU2001-285, pp. 105-112, 2002.

[2] 青木尊之, 額田彰, “はじめての CUDA プログラミング.” 株式会社工学社, 2009.

[3] 梅本英生, 佐々木太良, 成田清正, “フラクタル画像符号化のための色情報を用いた領域検索による計算量削減,” 2006 年電子情報通信学会基礎・境界ソサイエティ大会, page 130, 2006.