並列分散処理最終課題

195738E 知念弘也
2021年8月11日

1 目的

GPUの優位性を活かしたプログラムをCPUによる逐次処理とGPUによる並列処理で比較し、その効果を検証する。

2 方法

今回は、CUDAの強みが生かせる代表的な分野である画像処理を取り扱う予定であったが、私の環境で上手く動かない部分があったため、画像処理を模したプログラムを作成した。
今回行った実験の概要及び開発環境・実行環境は以下の通りである。

2.1 開発環境

Visual Studio Code 1.59.0

2.2 実行環境

CPU：AMD Ryzen 9 3900X
GPU：NVIDIA GeForce GTX 1650 (VRAM 4GB)
Windows 10 Pro (build 19042.1110)

・C用コンパイラ

gcc version 9.2.0 (MinGW.org GCC Build-2)

・CUDA実行にあたり必要なソフト

Visual Studio 2019 v16.10.3
CUDA Toolkit v11.4

2.3 リポジトリ

今回作成したソースコードはGithub上で公開している。

https://github.com/hiroya-ie/parallels2021_cuda

2.4 プログラムについて

今回は、カラー画像をモノクロ画像に変換する処理を配列によって疑似的に再現し、CPUでの処理とGPUでの処理を比較検証しようと考えた。
画像ファイルは幅×高さの数だけピクセルがあり、それぞれに対してRGB(各値0～255、8bit)が決められている、というのが基本的な形式である。一般的なJPEG形式などの圧縮形式はファイルサイズ削減のために全てのピクセルに対してRGB値を保持しているわけではないが、Windows bitmap(BMP)形式等はRGB値を保存しており、今回再現する処理が参考になると考えられる。
まず、今回は画像ファイルを使用しない。というのも、C言語での画像ファイルの読み込みには外部ライブラリを用いるか自作する必要があったため、環境により実行できなくなる可能性もあるためだ。また、今回行ったモノクロ画像への変換は、各ピクセルのRed、Green、Blueの平均値を出すことによる簡易的なものである。以下が今回使用した変数である。

・int width, height

プログラム実行後にキーボードからの入力により決定する。それぞれ画像の横と縦のピクセル数を表す。

・int cycle

プログラミング実行後にキーボードからの入力により決定する。1回のモノクロ変換では実行時間に差が生まれにくいため、for文で繰り返し実行することによりCPUとCUDAでの差をつけるようにしている。

・int pixel

width*heightにより求めた総ピクセル数である。この値をもとにRGBの各配列のメモリを確保する。

・unsigned char red,green,blue,out

今回は諧調を8bitとするため、0～255の値をとるunsigned charが適切だと判断した。

red、green、blueがそれぞれRGBに対応し、outはモノクロ化した際の各ピクセルの明度を格納する。

3 結果

実行時間の測定はclock関数を利用した。プログラム全体ではなく、Cは計算部分のみ、CUDAはデバイスのメモリ確保～計算～結果をホストへ転送するところまでである(コード参照)。また、私の実行環境に合わせてCUDAのブロック数512、スレッド数を1024としている。GPUによっては値を変更すると速度向上化みられるかもしれない。

今回は変数として入力するのはwidthとheightだが、速度にかかわるのはpixelとcycleであるため、この2値を変えて比較する。表の計測値は全て同様の処理を3回行った際の平均値である。

pixel	cycle	処理数	C	CUDA
1,000	10	10,000	0.000000s	0.133000s
100,000	10	1,000,000	0.002000s	0.136000s
1,000	1,000	1,000,000	0.002000s	0.134000s
1,000	10,000	10,000,000	0.020300s	0.130666s
10,000,000	10	100,000,000	0.220000s	0.142000s
100,000	1,000	100,000,000	0.204000s	0.128333s
1,000	100,000	100,000,000	0.202000s	0.128666s
100,000	10,000	1,000,000,000	2.047000s	0.137666s
10,000,000	1,000	10,000,000,000	21.839000s	0.395000s
100,000	100,000	10,000,000,000	20.909333s	0.283000s
10,000,000	10,000	100,000,000,000	222.872000s	11.344000s
10,000,000	100,000	1,000,000,000,000	計測不可	123.245333s

処理数の増大に伴ってCUDAによる並列処理の速さが目に見える結果となった。逐次処理であるCPUでの処理は、処理数に比例して計算時間が増えていった。それに反して並列処理であるCUDAでは、データの受け渡しによる遅延があるからか、処理数が1,000,000未満のときはCPUより遅かった。しかし、処理数が10,000,000以上になるとCPUより高速となった。検証した中で最も高速化が現れたのが処理数が1億のときで、CUDAはCPUに比べ74倍高速であった。また、処理数が同じであるとき、pixelとcycleの値の差が小さいほど高速になることが分かった。これはCPUもGPUも両方に言える。

また、検証の際に正常に動作しているか、printfでout値を出力するようにしたところ、きちんと計算されているように見受けられたため、動作に問題はないと思われる。

4 考察

今回の検証ではGPUの並列処理による高速化がが計測値として確認することができた。やはり、メモリの転送という部分でCUDAは不利であるため、処理数が少ない際はCPUの方が高速であった。ただ、処理中にGPUの共有メモリ等がどのように使用されているかが不明なため、チューニングの余地も十分にあると考えられる。また、今回はランダム関数を初期化していないため、その点に関して結果がどのように変化するか検証しても良いかもしれない。

5 感想

今回、初めてCUDAを使用したプログラム作成を行ったため、基本的なところでミスがあるかもしれないという心配があるなか、とりあえず高速化できているようだったので安心した。ただ、グリッド、ブロックやスレッド数に関しての最適化や共有メモリの有効利用に関しての知識も全くないため、更なる高速化の可能性があると思うと、GPUはCPUとは異なる方針で設計されていることを思い知らされた。今回行ったような画像処理(画像は用いていないが。)のような、独立性の高い値が膨大にある際はCUDAが活躍できるということを理解した。

6 役割分担

チーム23は私一人のため、全役割を担当した。

参考文献

[1]【C言語】mallocによる動的配列まとめ, mimizu, mimizublog, https://mimizublog.com/%E3%80%90c%E8%A8%80%E8%AA%9E%E3%80%91malloc%E3%81%AB%E3%82%88%E3%82%8B%E5%8B%95%E7%9A%84%E9%85%8D%E5%88%97%E3%81%BE%E3%81%A8%E3%82%81, 2021/08/11

[2] Tutorial 02: CUDA in Actions, Putt Sakdhnagool, CUDA Tutorial, https://cuda-tutorial.readthedocs.io/en/latest/tutorials/tutorial02/, 2021/08/11

[3] 2019/9/8 TensorflowをGPUで動かしてみた話, archangel_seraphy, eraphyの日記, https://seraphyware.wordpress.com/2019/09/08/2019-9-8-tensorflow%E3%82%92gpu%E3%81%A7%E5%8B%95%E3%81%8B%E3%81%97%E3%81%A6%E3%81%BF%E3%81%9F%E8%A9%B1/, 2021/08/11

[4] 繰り返し処理をCUDAで書く(配列同士の足し算), わざきた, Qiita, https://qiita.com/wazakkyd/items/8a5694e7a001465b6025, 2021/08/11