平成 28 年 実験 D
「CUDA による GPGPU の基礎」

淀川 滉也

2016 年 7 月 11 日

1 目的

CUDA を用いた一般的なプログラムの GPU 実装と速度評価を通して，並列処理および GPGPU の基礎を理解する．

2 原理

2.1 CUDA

CUDA のプログラミングモデルにおいて，CPU はホスト，GPU はデバイスと呼ばれる．GPU はビデオメモリ（デバイスメモリ）とストリーミングマルチプロセッサ (Streaming Multiprosessor:SM) を持ち，GPU における並列処理は SM を単位として行われる．CUDA のプログラムは拡張された C 言語を用いて記述され，CPU を動作させるホストコード，GPU を動作させるデバイスコードで構成される．ホストコードには，通常の C 言語で記述されたプログラムに加え，GPU で実行されるカーネル関数の呼び出しが含まれ，デバイスコードにはカーネル関数が記述される．

2.2 バイラテラルフィルタ

バイラテラルフィルタは，注目画素からの距離による重みと，注目画素との画素値の差に応じた重みを用いて平滑化を行うフィルタである．それぞれの重みはガウス分布に従う．注目画素からの距離のみに着目した平滑化を行うフィルタとしてガウシアンフィルタがあるが，ガウシアンフィルタでは原画像の輪郭がぼやけてしまう欠点がある．バイラテラルフィルタでは，注目画素との画素値の差に応じた重み付けも加えることで，上記の問題を改善している．

画像 $f(i,j)$ が与えられたとき，大きさ $(2w+1)$ のバイラテラルフィルタによって平滑化された画像 $g(i,j)$

は次式で与えられる．

$$g(i,j) = \frac{\sum_{n=-w}^{w} \sum_{m=-w}^{w} f(i+m, j+n) s(i,j,m,n)}{\sum_{n=-w}^{w} \sum_{m=-w}^{w} s(i,j,m,n)} \tag{1}$$

$$s(i,j,m,n) = \exp\left(-\frac{m^2+n^2}{2\sigma_1^2}\right) \exp\left(-\frac{(f(i,j)-f(i+m,j+n))^2}{2\sigma_2^2}\right) \tag{2}$$

ここで，σ_1, σ_2 はそれぞれ注目画素からの距離，注目画素との画素値の差それぞれについての重み付けに用いるガウス分布の分散である．

2.3 SSD によるテンプレートマッチング

テンプレートマッチングとは，テンプレートと画像が与えられたとき，画像中におけるテンプレートの位置を検出する処理である．テンプレートを画像全体に対して移動し，それぞれの位置で相違度を計算する．SSD (Sum of Squared Difference) による相違度 R_{SSD} は次式で与えられる．

$$R_{SSD} = \sum_{j=0}^{N-1} \sum_{i=0}^{M-1} \{I(i,j) - T(i,j)\}^2 \tag{3}$$

ここで，テンプレートの大きさを $M \times N$ 画素，テンプレートの位置 (i,j) における輝度値を $T(i,j)$，テンプレートと重ね合わせた対象画像の輝度値を $I(i,j)$ とする．相違度 R_{SSD} は，画像がテンプレートと完全に一致したとき 0 になり，相違が大きいほど大きな値となる．よって，R_{SSD} が最も小さくなるような位置を探索することで，対象画像中でのテンプレート位置を得ることができる．

3 実験

3.1 CUDA の動作確認

図 1に示すベクトルの加算を行うプログラムを実行し，その動作を確認する．

3.2 行列の積

2 つの $N \times N$ 行列の積を計算する C 言語プログラムを作成する．作成したプログラムを CUDA を用いて GPU に実装し，$N = 32, 64, 128, 256, 1024$ としたときの CPU 実装と GPU 実装の実行時間を計測し，比較する．また，GPU 実装についてはブロックサイズを $4 \times 4, 8 \times 8, 16 \times 16, 32 \times 32$ と変化させ，それぞれのブロックサイズで計測を行った．

3.3 画像の平滑化

バイラテラルフィルタを用いて画像を平滑化する C 言語プログラムを作成し，CUDA を用いて GPU に実装する．また，作成したプログラムについて，CPU 実装と GPU 実装の実行時間の計測・比較を行う．入力画像として，図 2に示す 1024×768 画素の画像を用いる．

3.4 テンプレートマッチング

SSD を用いてテンプレートマッチングを行い，相違度マップを出力する C 言語プログラムを作成し，CUDA を用いて GPU に実装する．また，作成したプログラムについて，CPU 実装と GPU 実装の実行時間の計測・比較を行う．マッチングに用いる画像とテンプレートとして，1024×768 画素の図3(a)と，64×64 画素の図3(b)を用いる．

4 実験結果・考察

4.1 CUDA の動作確認

図 1 のプログラムを実行した出力結果を表 1 に示す．結果より，2 つのベクトルの加算が正しく行われていることが確認できた．

4.2 行列の積

行列の積を求める C 言語のプログラム（CPU 実装）を図 4 に，図 4 のプログラムを CUDA を用いて GPU を利用するように変更したプログラム（GPU 実装）を図 5 に示す．

CPU 実装，GPU 実装のプログラムのそれぞれについて，行列のサイズ N を $N = 32, 64, 128, 256, 1024$, GPU 実装についてはブロックサイズを $1, 4, 8, 16, 32$ と変え，それぞれの条件での実行時間を記録した．結果は表 **??** のようになった．＃考察しよう＃

4.3 画像の平滑化

バイラテラルフィルタを用いて画像の平滑化を行うプログラムの CPU 実装と GPU 実装を，それぞれ図 $6, 7$ に示す．

CPU 実装，GPU 実装それぞれの実行時間を表 **??** に示す．＃考察＃

4.4 テンプレートマッチング

テンプレートマッチングを行うプログラムの，CPU 実装を図 8，GPU 実装を図 9 に示す．CPU 実装，GPU 実装それぞれのプログラムについて，いずれの実装でも $(582, 459)$ において相違度が最も小さくなった．また，それぞれの実装の実行時間は表 **??** のようになった．

```
#define VEC_SIZE 1024
#define BLOCK_SIZE 16
#include <stdio.h>

__global__ void VecAddKernel( float *, float *, float * );

int main() {
  int i;
  float *A = ( float* )malloc( sizeof( float ) * VEC_SIZE );
  float *B = ( float* )malloc( sizeof( float ) * VEC_SIZE );
  float *C = ( float* )malloc( sizeof( float ) * VEC_SIZE );

  for(i = 0; i < VEC_SIZE; i++){
    A[i] = rand();
    B[i] = rand();
  }

  float *d_A, *d_B, *d_C;
  cudaMalloc( (void**) &d_A, sizeof( float ) * VEC_SIZE );
  cudaMalloc( (void**) &d_B, sizeof( float ) * VEC_SIZE );
  cudaMalloc( (void**) &d_C, sizeof( float ) * VEC_SIZE );

  cudaMemcpy( d_A, A, sizeof( float ) * VEC_SIZE, cudaMemcpyHostToDevice );
  cudaMemcpy( d_B, B, sizeof( float ) * VEC_SIZE, cudaMemcpyHostToDevice );

  dim3 dimBlock( BLOCK_SIZE, 1 );
  dim3 dimGrid( VEC_SIZE / BLOCK_SIZE, 1 );
  VecAddKernel <<< dimGrid, dimBlock >>> ( d_A, d_B, d_C );

  cudaMemcpy(C, d_C, sizeof( float ) * VEC_SIZE, cudaMemcpyDeviceToHost);

  for(i = 0; i < VEC_SIZE; i++)
    printf("%d \t %f \t %f \t %f\n", i, A[i], B[i], C[i]);

  cudaFree(d_A);
  cudaFree(d_B);
  cudaFree(d_C);

  free(A);
  free(B);
  free(C);
```



```
}

__global__ void VecAddKernel( float *d_A, float *d_B, float *d_C ){
  int tid = blockIdx.x * blockDim.x + threadIdx.x;
```

図 2　平滑化する画像

表 1　ベクトルの加算結果

ベクトル A	ベクトル B	A+B
0.83	0.86	1.69
0.77	0.15	0.92
0.93	0.35	1.28

(a) 原画像

(b) テンプレート

図 3　テンプレートマッチングに用いる画像

```
#define N 1024

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>

void matmul(float *a, float *b, float *c);
double gettimeofday_sec();
void printmat(float *a);

int main(){

  float *A = ( float* )malloc( sizeof( float ) * N * N);
  float *B = ( float* )malloc( sizeof( float ) * N * N);
  float *C = ( float* )malloc( sizeof( float ) * N * N);

  int i, j;
  double t1;

  srand((unsigned)time(NULL));

  for(i = 0; i < N; i++){
    for(j = 0; j < N; j++){
      A[(i * N) + j] = ((float)(rand() % 100)) / 100.0;
      B[(i * N) + j] = ((float)(rand() % 100)) / 100.0;
    }
  }

  //printmat(A);
  // printmat(B);

  t1 = gettimeofday_sec();

  matmul(A, B, C);

  t1 = gettimeofday_sec() - t1;

  // printmat(C);

  printf("%f\n", C[1]);
  printf("%f\n", t1);

  free(A);
  free(B);
  free(C);
```

```
#define N 1024
#define BLOCK_SIZE 32

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>

__global__ void matmul(float *a, float *b, float *c);
double gettimeofday_sec();
void printmat(float *a);

int main(){

  float *A = ( float* )malloc( sizeof( float ) * N * N);
  float *B = ( float* )malloc( sizeof( float ) * N * N);
  float *C = ( float* )malloc( sizeof( float ) * N * N);

  int i, j;
  double t1;

  srand((unsigned)time(NULL));

  for(i = 0; i < N; i++){
    for(j = 0; j < N; j++){
      A[(i * N) + j] = (float)(rand() % 100) / 100.0;
      B[(i * N) + j] = (float)(rand() % 100) / 100.0;
    }
  }

  float *d_A, *d_B, *d_C;
  cudaMalloc((void**)&d_A, sizeof(float)*(N*N));
  cudaMalloc((void**)&d_B, sizeof(float)*(N*N));
  cudaMalloc((void**)&d_C, sizeof(float)*(N*N));

  cudaMemcpy(d_A, A, sizeof(float)*(N*N), cudaMemcpyHostToDevice);
  cudaMemcpy(d_B, B, sizeof(float)*(N*N), cudaMemcpyHostToDevice);
  cudaMemcpy(d_C, C, sizeof(float)*(N*N), cudaMemcpyHostToDevice);

  //printmat(A);
  //printmat(B);
```



```
  t1 = gettimeofday_sec();

  dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE);
  dim3 dimGrid(N/BLOCK_SIZE, N/BLOCK_SIZE);
```

```
#define FILTER_SIZE 3 //(FILTER_SIZE * 2) + 1
#define LOOP 1 //number of times of use of the filter
#define SIGMA1 10
#define SIGMA2 10

#define WIDTH 1024
#define HEIGHT 768
#define BLOCK_SIZE 16

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <sys/time.h>

void bilateral(float *img, float *out);
double gettimeofday_sec();



int main(){

  int i;
  double t;
  FILE *fp;

  float *img = (float*)malloc( sizeof(float) * WIDTH * HEIGHT);
  float *out = (float*)malloc( sizeof(float) * WIDTH * HEIGHT);

  fp = fopen("photo.dat", "r");
  for(i = 0; i < WIDTH * HEIGHT; i++){
    fscanf(fp, "%f", &img[i]);
  }
  fclose(fp);

  t = gettimeofday_sec();

  bilateral(img, out);

  t = gettimeofday_sec() - t;

  fp = fopen("out_cpu.dat", "w");
  for(i = 0; i < WIDTH * HEIGHT; i++){
    fprintf(fp, "%f\n", out[i]);
  }
  fclose(fp);
```

```
#define FILTER_SIZE 3 //(FILTER_SIZE * 2) + 1
#define LOOP 1 //number of times of use of the filter
#define SIGMA1 10
#define SIGMA2 10

#define WIDTH 1024
#define HEIGHT 768
#define BLOCK_SIZE 16

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <sys/time.h>

__global__ void bilateral(float *img, float *out);
double gettimeofday_sec();



int main(){

  int i;
  double t;
  FILE *fp;

  float *img = (float*)malloc( sizeof(float) * WIDTH * HEIGHT);
  float *out = (float*)malloc( sizeof(float) * WIDTH * HEIGHT);

  fp = fopen("photo.dat", "r");
  for(i = 0; i < WIDTH * HEIGHT; i++){
    fscanf(fp, "%f", &img[i]);
  }
  fclose(fp);

  float *d_img, *d_out;
  cudaMalloc((void**)&d_img, sizeof(float)* WIDTH * HEIGHT);
  cudaMalloc((void**)&d_out, sizeof(float)* WIDTH * HEIGHT);

  cudaMemcpy(d_img, img, sizeof(float)* WIDTH * HEIGHT, cudaMemcpyHostToDevice);
  cudaMemcpy(d_out, out, sizeof(float)* WIDTH * HEIGHT, cudaMemcpyHostToDevice);

  dim3 dimBlock(BLOCK_SIZE, BLOCK_SIZE);
  dim3 dimGrid(WIDTH/BLOCK_SIZE, HEIGHT/BLOCK_SIZE);

  t = gettimeofday_sec();
```

```
#define IMAGE_WIDTH 1024
#define IMAGE_HEIGHT 768
#define TMP_SIZE 64


#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>


double gettimeofday_sec(){

  struct timeval tv;
  gettimeofday(&tv, NULL);
  return tv.tv_sec + (double)tv.tv_usec*1e-6;

}


int main(){

  int i, j, k, l;
  int *img = (int*)malloc(sizeof(int) * IMAGE_WIDTH * IMAGE_HEIGHT);
  int *tmp = (int*)malloc(sizeof(int) * TMP_SIZE * TMP_SIZE);

  int *Rssd = (int*)malloc(sizeof(int) * (IMAGE_WIDTH - TMP_SIZE + 1) * (IMAGE_HEIGHT - TMP_SIZE + 1
  int x;

  int r = 10000;
  int minx = 0;
  int miny = 0;

  FILE *fp;

  fp = fopen("image.dat", "r");
  for(i = 0; i < IMAGE_WIDTH * IMAGE_HEIGHT; i++){
    fscanf(fp, "%d", &img[i]);
  }
  fclose(fp);

  fp = fopen("template.dat", "r");
  for(i = 0; i < TMP_SIZE * TMP_SIZE; i++){
    fscanf(fp, "%d", &tmp[i]);
  }
```

```
#define IMAGE_WIDTH 1024
#define IMAGE_HEIGHT 768
#define TMP_SIZE 64
#define BLOCK_SIZE 16

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>


double gettimeofday_sec(){

  struct timeval tv;
  gettimeofday(&tv, NULL);
  return tv.tv_sec + (double)tv.tv_usec*1e-6;

}


__global__ void tmpmatch(int *img, int *tmp, int *Rssd){
  int j = blockIdx.x * blockDim.x + threadIdx.x;
  int i = blockIdx.y * blockDim.y + threadIdx.y;

  int k, l;

  int x = 0;

  for(k = 0; k < TMP_SIZE; k++){
    for(l = 0; l < TMP_SIZE; l++){
      x += (img[(k+i) * IMAGE_WIDTH + (l+j)] - tmp[k * TMP_SIZE + l]) *
(img[(k+i) * IMAGE_WIDTH + (l+j)] - tmp[k * TMP_SIZE + l]);
    }
  }

  Rssd[i * (IMAGE_WIDTH - TMP_SIZE + 1) + j] = x;


}

int main(){
```



```
  int i, j;
  int *img = (int*)malloc(sizeof(int) * IMAGE_WIDTH * IMAGE_HEIGHT);
  int *tmp = (int*)malloc(sizeof(int) * TMP_SIZE * TMP_SIZE);
  int *Rssd = (int*)malloc(sizeof(int) * (IMAGE_WIDTH - TMP_SIZE + 1) * (IMAGE_HEIGHT - TMP_SIZE + 1
```