Christos Axelos, AEM 1814, 5th Assignment in High Performance computing 2017/18

OBSERVATIONS

- Πειραματίστηκα στην εικόνα με το μεγαλύτερο μέγεθος, την **planet_surface.pgm**, όπου το μέγεθος της είναι **6400 x 6400**

ANALYSIS

- Στον **ακολουθιακό** κώδικα, ο **μέσος χρόνος** εκτέλεσης και η **διακύμανση** για την εικόνα εισόδου 6400x6400 είναι 0.19422 secs και 0.000385042713secs αντίστοιχα.

- Αρχικά έτρεξα το πρόγραμμα μέσα από το **vtune** για να εντοπίσω τα σημεία όπου πρέπει αρχικά να εστιάσω. Μέσω του vtune έβγαλα τα εξής συμπεράσματα:

- Το **41.2%** του συνολικού χρόνου εκτέλεσης ξοδεύεται στην συνάρτηση **histogram_equalization()**, συγκεκριμένα:

a) Το **16.5%** του χρόνου ξοδεύεται στην εντολή img_out[i] = (unsigned char)lut[img_in[i]];

b) Το **11.8%** του χρόνου ξοδεύεται στην εντολή img_out[i] = 255;

c) Το **8.2%** του χρόνου ξοδεύεται στην εντολή if(lut[img_in[i]] > 255)

δ) Το υπόλοιπο **4.7%** του χρόνου ξοδεύεται σε μοιρασμένα σημεία, οποτέ δεν μας απασχολεί προς το παρόν

ε) Επιπλέον, 11.8% του χρόνου ξοδεύεται στην εντολή hist_out[img_in[i]] ++; στην συνάρτηση **histogram()**

*1η Προσπάθεια: Βελτιστοποίηση συνάρτησης **histogram**() με χρήση Cuda με χρήση **μερικών αθροισμάτων***

- Ο αρχικός κώδικας της συνάρτησης histogram είναι ο εξης:

```
void histogram(int * hist_out, unsigned char * img_in, int img_size, int nbr_bin){
    int i;
    for ( i = 0; i < nbr_bin; i ++){
        hist_out[i] = 0;
    }

    for ( i = 0; i < img_size; i ++){
        hist_out[img_in[i]] ++;
    }
}
```

- Στο 1ο loop αρχικοποιούμε το Histogramm με μηδέν ενώ στο 2ο loop αυξάνουμε κατά την θέση(από 0 μέχρι 255) που προκύπτει από το img_in[i].

- Επειδή το 2ο Loop εκτελείται **IMG_WIDTH * IMG_HEIGHT** φορές, παραλληλοποιώ την συνάρτηση ώστε να εκτελείται όσο πιο παράλληλα γίνεται. Στη συγκεκριμένη περίπτωση, θα είναι **6400x6400** για την εικόνα planet_surface.pgm

- Χρησιμοποιώ την τεχνική του **partial-sum** όπου δημιουργώ NUM_OF_BLOCKS int[256] πίνακες, που ο καθένας συνεισφέρει στον τελικό int[256] πίνακα

- Χρησιμοποιώ **32x32 threads** για κάθε block, ενώ στην περίπτωση της εικόνας 6400 x 600, χρησιμοποιώ grid των 200x200 blocks. Άρα σύνολο **40000 blocks**

- Ο κάθε πίνακας int[256] είναι δεσμευμένος στην **shared memory** του κάθε block, ενώ τα 32x32 threads του κάθε block συνεισφέρουν σαυτόν

- Εκτελώ τον kernel histogram<<<numBlocks, threadsPerBlock>>>(d_partial_hist, d_img_in);

- Τα μερικά αθροίσματα του Kernel τα αποθηκεύω στον πίνακα **d_hist_out**, μεγέθους 40000 * 256 * sizeof(int)

- Ο κώδικας του kernel φαίνεται στην **εικόνα 1.1**

```
__global__ void histogram(int * d_hist_out, unsigned char * img_in){

    unsigned long global_index = (blockIdx.y*blockDim.y + threadIdx.y) * (gridDim.x * blockDim.x)
                               + ( blockIdx.x*blockDim.x + threadIdx.x );

    int thread_index = threadIdx.y * blockDim.x + threadIdx.x;

    __shared__ int sh_hist_out[256];


    if ( thread_index < 256)  {
          sh_hist_out[thread_index] = 0;
    }
    __syncthreads();


    atomicAdd( &sh_hist_out[ img_in[global_index] ], 1);// ++;

    __syncthreads();

    if ( thread_index < 256 )   {
        d_hist_out[( blockIdx.y*gridDim.x + blockIdx.x)*256 + thread_index] = sh_hist_out[thread_index];
    }

}
```

(εικόνα 1.1)

- Μόλις ο kernel εκτελεστεί ο Kernel, διαχειρίζομαι τα μερικά αθροίσματα με τον εξης τρόπο**(εικόνα 1.2)**

```
    dim3 threadsPerBlock(32,32);
    dim3 numBlocks ( 6400/threadsPerBlock.x, 6400/threadsPerBlock.y );

    histogram<<<numBlocks, threadsPerBlock>>>(d_partial_hist, d_img_in);//, img_in.h * img_in.w);//, 256);

    cudaThreadSynchronize();//barrier of host

    cudaMemcpy(h_partial_hist, d_partial_hist,40000*256*sizeof(int), cudaMemcpyDeviceToHost);

    for (j = 0; j < 40000; j++)  {
      for (i = 0; i < 256; i++)  {
        hist[i] = hist[i] + h_partial_hist[j*256 + i];
      }
    }

     CUDA_ERROR_CHECK(4);
    cudaFree( d_partial_hist);
    cudaFree( d_img_in );
```

(εικόνα 1.2)

- Προσθέτω τα μερικά αθροίσματα στον τελικό πίνακα, ώστε για κάθε μία θέση του τελικού πίνακα, να υπάρχουν **40000 συνεισφορές**

- Ο χρόνος στην 1η προσπάθεια παραλληλοποίησης είναι χειρότερος από την σειριακή εκτέλεση, αφού ο συνολικός χρόνος υπολογισμού είναι **0.25485** seconds, με απόκλιση 0.01582743seconds. Από αυτόν τον χρονο, οι καθυστερήσεις που προκαλεί η παραλληλοποίηση είναι:

a) Χρόνος **μεταφορών** προς την μνήμη του Device: 0.15896seconds με απόκλιση 0.0158407seconds

b) Χρόνος **μεταφορών** από το Device προς την μνήμη: 0.01006 seconds με απόκλιση 0.0158407seconds

c) Χρόνος **εκτέλεσης** του Kernel: 0.00003 seconds με απόκλιση 0.0000006741 seconds

d) Χρόνος υπολογισμού του τελικού histogram: 0.00271 seconds με απόκλιση 0.00005641 seconds