علی عسگری
40032223

تمرین سوم
تکمیلی
پیاده سازی با اندکی تفاوت

علاوه بر فایل matrix_sum که در ارسال قبلی توضیحات آن ارائه شد ، این بار فایل حاوی کد کودا با نام mat_sum ایجاد شده و یک تفاوت اساسی با نحوه کد نویسی قبلی دارد.
بقیه قسمت ها مشابه روش قبلیست پس صرفا به بیان این تفاوت و اختلاف میپردازم.

```
donalito@DESKTOP-HJLUASU:/mnt/c/Users/ALI/Downloads/tamrins/t1_algo_parallel/3/08-intro-to-cuda$ ./mat_sum 2 2
m = 2, n = 2
Enter some numbers and we create 2 matrix , A and B , first A :
2
2
2
2
Enter second matrix: 2
22
2
2
A =
2.0 2.0
2.0 2.0
B =
2.0 22.0
2.0 2.0
The sum is:
0.0 0.0
0.0 0.0
```

همانطور که مشهود است خروجی این روش با خروجی روش قبلی از لحاظ کارکرد و محاسبات یکسان است.

اما اختلاف روش دوم و روش اول در بحث فراخوانی کرنل GPU است ، به این صورت که به جای ایجاد تعدادی بلاک و در واقع به جای تثبیت ابعاد یک بلاک و به دست آوردن تعداد بلاک ها و در نهایت جای گذاری آنها در فرآیند فراخوانی کرنل ، از ابعاد ماتریس برای فراخوانی کرنل استفاده میکنیم.

```
/* Invoke kernel using m thread blocks, each of    */
/* which contains n threads                        */
Mat_sum<<<m, n>>>(d_A, d_B, d_C, m, n);

/* Wait for the kernel to complete */
cudaThreadSynchronize();

/* Copy result from device memory to host memory */
```

در قطعه کد بالا m و n در واقع ابعاد ماتریس ما هستند ، میتوان اینطور فرض کرد که یک بلاک داریم که ابعادش با ابعاد ماتریس برابر است.

```
o-to-cuda$ ./mat_sum 2 2
```

```c
/* Get size */
if (argc != 3) {
    fprintf(stderr, "usage: %s <row count> <col count>\n", argv[0]);
    exit(0);
}
m = strtol(argv[1], NULL, 10);
n = strtol(argv[2], NULL, 10);
printf("m = %d, n = %d\n", m, n);
size = m*n*sizeof(float);

h_A = (float*) malloc(size);
h_B = (float*) malloc(size);
h_C = (float*) malloc(size);
```

نحوه دریافت و نمایش m و n هم در تصویر بالا مشهود است ، همچنین نحوه محاسبه میزان فضای مورد نیاز و ذخیره آن در متغیر size هم مشهود است ، در ادامه به تخصیص فضا پرداخته شده است.