نگین فیروزیان 9431018

گزارش پروژه­ی پایانی

**تابع main()**

int main(int argc, char \*\*argv)

{

//char \*\*strings1 = (char\*\*)malloc(10 \* sizeof(char\*));

char arr1[10][30];

FILE \* database;

char buffer1[30];

int Count1 = 0;

database = fopen("SongNames.txt", "r");

if (NULL == database)

{

perror("opening database");

return (-1);

}

while (EOF != fscanf(database, "%[^\n]\n", buffer1))

{

//printf("> %s\n", buffer1);

strcpy(arr1[Count1], buffer1);

Count1++;

}

fclose(database);

char arr2[10][30];

FILE \* database2;

char buffer2[30];

int Count2 = 0;

database2 = fopen("SampleNames.txt", "r");

if (NULL == database2)

{

perror("opening database");

return (-1);

}

while (EOF != fscanf(database2, "%[^\n]\n", buffer2))

{

//printf("> %s\n", buffer2);

strcpy(arr2[Count2], buffer2);

Count2++;

}

fclose(database2);

for (int i = 0; i < Count1; i++) {

for (int j = 0; j < Count2; j++) {

double a = 0.0;

char path1[30]="songs/";

char path2[30] = "samples/";

concatenate\_string(path1, arr1[i]);

concatenate\_string(path2, arr2[i]);

CompareWav(path1, path2,&a);

printf("\%s >>> %s : Similarity rate: %f\n", arr1[i], arr2[j], a);

}

}

}

در این تابع ابتدا فایل های دورن پوشه های song و sample خوانده می­شوند و در یک آرایه ذخیره می­شوند سپس به ازای هر جفت فایل از پوشه­ی song و sample تابع کودا صدا زده می­شود و در پایان عددی به عنوان درجه­ی تفاوت آن دو فایل چاپ می­شود که مینیمم LAD بین دو فایل در حالت های مختلف مقایسه را نشان می­دهد.

**تابع CompareWav**

ابتدا آرایه های host را صدا زدن تابع readFile مقدر دهی می­کنیم. این تابع آرایه اعداد خوانده شد از فایل را داخل آرایه­های host میریزد.

پس از ایجاد آرایه های host و device از کتابخانه cuFFT برای تبدیل فوریه استفاده می­شود.

cufftHandle planA, planB;

cufftPlan1d(&planA, count\_A, CUFFT\_C2C, 1);

cufftPlan1d(&planB, count\_B, CUFFT\_C2C, 1);

// Transform signal and kernel

printf("Transforming signal cufftExecC2C\n");

cufftExecC2C(planA, (cufftComplex \*)d\_A, (cufftComplex \*)d\_A, CUFFT\_FORWARD);

cufftExecC2C(planB, (cufftComplex \*)d\_B, (cufftComplex \*)d\_B, CUFFT\_FORWARD);

سپس تابع کرنل فراخوانی می­شود که در ادامه توضیح داده خواهد شد و پس از فراخوانی کرنل مقادیر خروجی device درون host کپی می­شود و تابع با آزاد کردن متغیر ها خاتمه می­یاد.

**تابع کرنل compareKernel**

در این تابع A و B به عنوان ورودی ها هستند و A آهنگ اصلی و B سمپل آهنگ است. الگوریتم کلی به این صورت است که به میزان اختلافی که سایز A و B با هم دارند نیاز است که با هم مقایسه شوند. حلقه­ی for اول برای هندل کردن تعداد نخ های کم تر از سایز A است. داخل این حلقه به تعداد (sizeA - sizeB) + 1 نخ استفاده می­شود تا هر کدام از خانه A[threadID] تا خانه­ی A[threadID + sizeB -1] را با آرایه B مقایسه کند و مقدار LAD آن را محاسبه کند. در نهایت کوچکترین LAD پیدا می­شود و به عنوان آخرین خانه­ی آرایه­ی C به هاست ارسال می­گردد.

const int numThreads = blockDim.x \* gridDim.x;

const int threadID = blockIdx.x \* blockDim.x + threadIdx.x;

double valueA = 0.0;

double valueB = 0.0;

double diff = 99999999999999999999999.9;

//double fab[sizeA];

//int i = threadID;

for (int i = threadID; i < ((sizeA- sizeB)+1); i += numThreads) {

if ((i + (sizeB - 1)) < sizeA) {

res[i] = 0.0;

for (int j = 0; j < sizeB; j++) {

res[i] += fabs(sqrt(pow(A[i + j].x, 2) + pow(A[i + j].y, 2)) - sqrt(pow(B[j].x, 2) + pow(B[j].y, 2)));

}

if (res[i] < diff) {

diff = res[i];

}

}

res[(sizeA - sizeB) + 1] = diff;

}

\*فایل cufft64\_91.dll به دلیل حجم بسیار بالا در فایل­ها قرار داده نشده است اما برای اجرای برنامه لازم است.