به نام خدا

پروژه درس ساختار و زبان کامپیوتر

ترم1-1404

سعید نوفرستی 403106838

-هدف این پروژه پیاده‌سازی و مقایسه‌ی **کانولوشن دوبعدی تصویر (2D Convolution)** به دو روش است:

۱. پیاده‌سازی ساده با C (Baseline)
۲. پیاده‌سازی بسیار سریع با Assembly + AVX SIMD

سپس:
- مقایسه زمان اجرا (Benchmark)
- محاسبه Speedup
- Zero Padding
- تشخیص لبه با Sobel
- تشخیص شی (Object Recognition)
-تشخیص الگو (Pattern Recognition)
- اجرای خودکار روی صدها تصویر

این برنامه روی تصاویر pgm کار میکند پس ابتدا نیاز داریم این دستور را در ترمینال لینوکس وارد کنیم

```
sudo apt update
sudo apt install imagemagick
```

سپس هر تصویر دلخواه را با دستور

```
convert input.jpg -colorspace Gray input.pgm
```

به تصویر pgm تبدیل میکنیم

ساختار کلی برنامه به اینصورت است

```
├── main.c                  # فایل اصلی برنامه
├── conv.c                  # نسخه C کانولوشن
├── conv.asm                # نسخه Assembly + AVX
├── conv.h                  #هدر
├── timer.c                 # اندازه‌گیری زمان
├── object_recognition.c    # پردازش 400 تصویر و تشخیص شی
├── inputs/                 # تصاویر ورودی
├── Makefile                # پیکر بندی اجرایی
└── pattern_recognition.c  #پردازش تصاویر و تشخیص الگو
```

فایل c برای محاسبه تایم(time.c)

timer.c

```c
#include <time.h>

double now(){
    struct timespec ts;
    clock_gettime(CLOCK_MONOTONIC, &ts);  // گرفتن تایم از MONOTONIC
    return ts.tv_sec + ts.tv_nsec*1e-9; // برگرداندن تایم به واحد ثانیه
}
```

فایل پیکر بندی برای اجرای راحت تر برنامه (Makefile)

Makefile

```
# کامپایلر C
CC=gcc

# اسمبلر NASM
ASM=nasm

# فلگهای کامپایل:
# -O3 : بیشترین سطح بهینه‌سازی
# -mavx : فعال‌سازی دستورات AVX
CFLAGS=-O3 -mavx


# تارگت پیش‌فرض -> ساخت فایل اجرایی main
all: main


# تبدیل فایل اسمبلی به آبجکت
conv.o: conv.asm
    $(ASM) -f elf64 conv.asm -o conv.o


# کامپایل و لینک کل پروژه
# -lm برای توابع ریاضی مثل fabs,...
main: main.c conv.c  object_recognition.c timer.c conv.o
    $(CC) $(CFLAGS) main.c conv.c object_recognition.c timer.c conv.o -lm -o main



# پاک کردن فایل‌های ساخته شده
clean:
    rm -f main *.o output_c.pgm output_asm.pgm
```

کرنل های متناسب با فیلتر های مختلف

```
// Edge_Detection
float ker_Edge[9] = {
    -1,-1,-1,
    -1, 8,-1,
    -1,-1,-1
};

// Blur
float ker_Blur[9] = {
    1.00f/9.00f,1.00f/9.00f,1.00f/9.00f,
    1.00f/9.00f,1.00f/9.00f,1.00f/9.00f,
    1.00f/9.00f,1.00f/9.00f,1.00f/9.00f
};

// Sharpen
float ker_sharpen[9] = {
0, -1,  0,
-1,  5, -1,
0, -1,  0
};
```

فایل conv.h برای مشخص کردن پروتوتایپ تابع conv2d به زبان c و asm

```
13
14  void conv2d_c(float* in, float* out, float* kernel,
15                int w, int h, int k);
16
17
18  void conv2d_asm(float* in, float* out, float* kernel,
19                  int w, int h, int k);
20
```

کد توابع در فایل های conv2d.c و conv2d.asm موجود است حسن asm نسبت به c استفاده از AVX و توابع برداری است به عنوان مثال در تصویر زیر نمونه ان را میبینیم :

```
vbroadcastss ymm2, [rdx + rax*4] ;خواندن وزن از کرنل

; خواندن 8 پیکسل از تصویر: image[((y+ky-pad)*w) + (x+kx-pad)]
mov rax, rbx  ; rax = y = rbx
add rax, r13   ; rax = y + ky
sub rax, r14    ;rax = y + ky - pad
imul rax, rcx    ;rax = (y + ky - pad) * w -> سطر
add rax, r12        ; اضافه کردن x
add rax, r10        ; اضافه کردن kx
sub rax, r14        ; کسر کردن pad
;rax = iy * w - ix

vmovups ymm1, [rdi + rax*4] ;خواندن از ارایه ورودی

;  ضرب و جمع در یک سیکل پرداشی
vfmadd231ps ymm0, ymm1, ymm2 ; ymm0 = (ymm1 * ymm2) + ymm0
```

فایل object_recognition.h برای مشخص کردن پروتوتایپ تابع object_recognition.c که برای پردازش چند صد تصویر به زبان asm, c و تشخیص شی استفاده میشود, است

object_recognition.h

```
1  #pragma once // جلوگیری از چندبار include (جلوگیری از خطای تکرار تعریف) شدن فایل هدر
2  // اجرای تشخیص شی روی چند تصویر
3  // folder : پوشه تصاویر (مثلاً inputs)
4  // count  : تعداد تصاویر
5  void objectRecognition(const char* folder, int count);
6
```

حالا به سراغ اجرای برنامه روی تصویر دلخواه میرویم
این برنامه پنج مد کاری دارد

۱. blur (تک تصویره)

۲. sharpen (تک تصویره)

۳.Edge_Detection (تک تصویره)

۴.object recognition (روی ۴۰۰ تصویر دارای شی یا بدون شی)
۵. pattern_recognition (تشخیص الگو)

برای شروع یک تصویر(pgm) را انتخاب میکنم برای مثال تصویر زیر:

1. اجرای فیلتر blur

```
make clean
make
./main Blur input30.pgm
```

خروجی ASM و C :

که به وضوح نسبت به تصویر اول blur شده است

خروجی کد در ترمینال به این شکل بود:

```
saeed@saeed-Vivobook-ASUSLaptop-K3605VU-K3605VU:~/me/Convolution$ ./main Blur input30.pgm
C time = 0.005142
ASM time = 0.000719
speedup : 7.153068
error : 0.000000
```

که نشان میدهد سرعت کار با اسمبلی 7.15 برابر کدC است و ارور صفر نشان دهنده این است که خروجی asmوc دقیقا مشابه هم هست با این تفاوت که سرعت asm بسیار بیشتر بوده است

2.اجرای فیلتر Sharpen

حالا تصویر blur شده در قسمت قبل با بار نام input.pgm ذخیره کرده و روی ان فیلتر sharpenرا اجرا میکنیم:

```
make clean
make
./main Sharpen input.pgm
```

اینبار خروجی در ترمینال به این شکل بود

```
saeed@saeed-Vivobook-ASUSLaptop-K3605VU-K3605VU:~/me/Convolution$ ./main Sharpen input.pgm
C time = 0.005701
ASM time = 0.000936
speedup : 6.088460
error : 0.000000
```

و خروجی تصویر به شکل زیر است که به وضوح از تصویر blurشده قبل شارپ تر شده است

C

ASM

3.اجرای فیلتر Edge_Detection

```
make clean
make
./main Edge_Detection input30.pgm
```

خروجی کد

```
saeed@saeed-Vivobook-ASUSLaptop-K3605VU-K3605VU:~/me/Convolution$ ./main Edge_Detection input30.pgm
C time = 0.004440
ASM time = 0.000866
speedup : 5.127548
error : 0.000000
```

تصویر خروجی

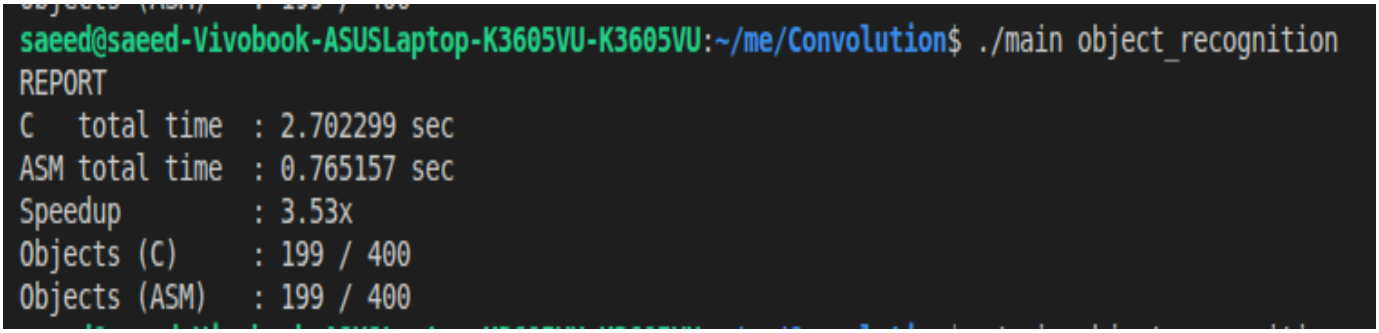
ASM

C

لبه های تصویر مشخص شده هست

-حالا سراغ تشخیص شی در ۴۰۰ تصویر مختلف میرویم اینبار در پوشه inputs تصاویر مختلف که ۲۰۰ تصویر دارای شی و ۲۰۰ تصویر تصاویر سفید و خاکستری و مشکی و... بدون شی هستند را قرار میدهیم تا روی این تصاویر این ازمایش را انجام دهیم

make clean
make
./main object_recognition
خروجی کد به این صورت بود



حتی با وجود اینکه در تایم اسمبلی تایم zero paddingهم محاسبه شده باز هم حدود 3.5برابر سرعت بیشتری به ما میدهد که شگفت انگیز است
اما میبینیم که خروجی از لحاظ تشخیص تعداد تصویر برای هر دو کد c , asm یکسان است
دقت محاسبه به این صورت است که :
۱ تصویر که دارای شی بوده به اشتباه بدون شی تشخیص داده شده و ۳۹۹ تصویر درست تشخیص داده شده است:
پس با دقت 399/400*100 درصد محاسبه انجام شده

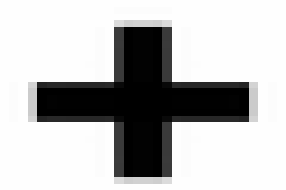
یعنی دقت برابر بوده با :

% 99.75

-بخش بعدی که بخش اصلی است بخش تشخیص الگوست
در این بخش یک الگو به عنوان کرنل به برنامه داده میشود
در ابتدا با این دستورات کرنل مربوطه را به فرمت استاندارد میرسانیم

```
convert ker.jpg/png -colorspace Gray ker.pgm
convert ker.pgm -resize 30x30 ker.pgm
```

در فولدر دلخواه تصاویر شامل الگو و تصاویری که شامل الگو نیستند را میگذاریم ۱۰۰ تصویر به عنوان تصویر بدون الگو و ۵۰ تصویر شامل الگو
به عنوان مثال الگو مورد نظر را علامت (+) قرار میدهیم با کرنل ۳۰ در ۳۰ زیر :



حالا کافی است دستور زیر را اجرا کنیم :

```
make clean
make
./main pattern_recognition kernels/ker.pgm
```

به علت اینکه کرنل ۳۰ در ۳۰ بود و به علت محاسبه NCC زمان اجرا بالا رفت اما با وجود زیرو پدینگ در اسمبلی باز همحدود ۲.۵ برابر سرعت گرفتیم از اسمبلی و ۴۹ تصویر دارای این الگو تشخیص داده شد که با بررسی نام تصاویری که دارای این الگو تشخیص داده شده اند متوجه میشویم که از این تعداد ۴۲ تصویر درست شناسایی شده بودند

۹۰٪ = ۱۰۰ * (۴۲+۹۳)/ ۱۵۰

از طرفی توجه داریم که خروجی های کد سی و اسمبلی دقیقا مشابه هم هست.

-حالا به بررسی اینکه کد pattern_recognition چطور کار میکند میپردازیم :
معیار و ملاک کد ما برای اینکه بگوید الگو پیدا شد بر اساس یک مفهوم ریاضی به نام
Normalized Cross-Correlation (NCC) یا «همبستگی متقاطع نرمال‌شده» است.

۱. مرحله‌ی تطبیق

وقتی تابع اسمبلی اجرا می‌شود، پیکسل‌های الگو (کرنل) را روی پیکسل‌های تصویر ضرب می‌کند.

- **اگر الگو منطبق باشد**: پیکسل‌های روشن الگو در پیکسل‌های روشن تصویر ضرب می‌شوند (عدد بزرگ * عدد بزرگ) و یک مقدار بسیار بزرگ تولید می‌کنند.
- **اگر منطبق نباشد**: اعداد بزرگ در اعداد کوچک (نزدیک به صفر) ضرب می‌شوند و خروجی ناچیزی می‌دهند.

مشکل اینجاست: اگر فقط به این عدد بزرگ اکتفا کنیم، یک دیوار کاملاً سفید هم می‌تواند عدد بزرگی تولید کند و کد را فریب دهد برای همین به مرحله دوم نیاز داریم

۲. مرحله‌ی نرمال‌سازی (Normalization) - "ملاک اصلی"

. ما مقدار خروجی را بر «**انرژی**» آن بخش از تصویر تقسیم می‌کنیم.

- **انرژی الگو**: یعنی چقدر پیکسل‌های سفید مثلا در علامت + داریم.
- **انرژی پنجره تصویر**: یعنی آن تکه از تصویر چقدر روشن است.

معیار واقعی این است کد بررسی می‌کند که آیا این درخشش به خاطر «**شکل و ساختار**» الگو است یا فقط به خاطر «**روشنایی زیاد**» آن تکه از تصویر. فرمول NCC خروجی را همیشه بین **۰ تا ۱** (یا ۰ تا ۱۰۰ درصد) نگه می‌دارد. عدد ۱ یعنی این تکه از تصویر از نظر ساختاری مثلا همان علامت + است.

چرا NCC

"چون NCC نسبت به تغییرات روشنایی تصویر و کنتراست مقاوم است. این فرمول شباهت را نه بر اساس «قدرت پیکسل‌ها»، بلکه بر اساس «هم‌بستگی هندسی و ساختاری» می‌سنجد. با نرمال‌سازی خروجی در بازه [۰, ۱]، توانستیم یک آستانه (Threshold) دقیق برای تشخیص قطعی الگو تعریف کنیم."

```
    //  محاسبه انرژی خودِ الگو (Template Energy)
    energy_k = 0;
    for(int i = 0; i < w_k * h_k; i++) {
        energy_k += ker[i] * ker[i];
    }

    max_ncc = -1.0f;
    for (int y = 0; y < h - h_k; y++) {
        for (int x = 0; x < w - w_k; x++) {
            //  محاسبه انرژی تکه‌ای از تصویر که زیر الگو قرار دارد (Window Energy)
            float energy_win = 0;
            for (int ky = 0; ky < h_k; ky++) {
                for (int kx = 0; kx < w_k; kx++) {
                    float pixel = in[(y + ky) * w + (x + kx)];
                    energy_win += pixel * pixel;
                }
            }

            //  مقدار خروجی کانولوشن در این نقطه
            float val = out_asm[y * w + x];

            //  فرمول NCC: همبستگی تقسیم بر جذر ضرب انرژی‌ها
            float denom = sqrtf(energy_k * energy_win);
            float ncc = (denom > 0) ? (val / denom) : 0;

            if (ncc > max_ncc) {
                max_ncc = ncc;
            }
        }
    }
```

محاسبه انرژی الگو و NCC

۳. مرحله‌ی تصمیم‌گیری (The Threshold)

در نهایت، کد به یک عدد (Confidence) می‌رسد.

اگر بالای ۸۵٪ بود: یعنی با احتمال خوبی الگو را تایید می‌کند.

اگر پایین ۸۵٪ بود: یعنی شاید شباهت‌هایی وجود داشته باشد (مثلاً یک خط عمودی پیدا شده)، اما چون شبیه به «کلِ» مثلا علامت + نیست، آن را رد می‌کند.

```
confidence = max_ncc * 100.0f;
if (confidence < 0) confidence = 0;
if (confidence > 100) confidence = 100;
if (confidence > 85.0f) { // با دقت 85 درصد
    patternFounded_asm ++;
}
```

این پروژه در لینک زیر در گیت هاب هم دردسترس است

https://github.com/SSN4444/Convolution.git

با تشکر از استاد گرامی دکتر جهانگیر