



پروژه درس ساختار و زبان ماشین

دکتر جهانگیر

دانشجو : امیرکسری احمدی «401170507»

زمستان ۱۴۰۲

سوال : برنامه اسمبلی ضرب دو ماتریس $n \times n$ اعداد ممیز شناور 32 بیتی به روش معمولی، یعنی ضرب و جمع متوالی درایه‌های متناظر دو ماتریس و مقایسه سرعت اجرای این برنامه با استفاده از دستورات برداری (موازی) پردازنده را برای پردازنده $80x86$ در دو حالت $n=3$ و $n=5$ بنویسید و با هم مقایسه کنید. همین طور با زمان اجرای برنامه‌ی ضرب ماتریس به زبان سطح بالا مقایسه و نقد کنید.

دو برنامه اسمبلی نوشتم که در یکی از آن‌ها از روش معمولی ضرب دو ماتریس را انجام می‌دهم و به این شکل است که در هر مرحله ضرب در ۳ تا حلقه تو در تو است که در هر بار پیمایش این حلقه درونی تنها یکی از درایه‌های ماتریس‌ها در هم ضرب می‌شوند و همان حالت معمولی ضرب دو ماتریس است که کد آن در فایل `cross_normal.asm` موجود است.

برنامه دیگر ما به این شکل است که در هر مرحله ضرب چهار عدد از درایه‌های ماتریس در هم ضرب می‌شوند و با هم جمع می‌شوند (ضرب داخلی دو آرایه که هر کدام ۴ درایه دارند) . به همین خاطر نیاز است که درایه‌های نهایی این دو آرایه اگر مضرب ۴ نباشد ۰ قرار بگیرند که مقدار درست برگرداند به همین خاطر ابتدا من مقدار `number` خود را به نزدیک‌ترین مضرب ۴ می‌رسانیم که و ماتریس خود را به اصطلاح `expand` می‌کنیم و باقی درایه‌ها را صفر می‌گذاریم.

حالا نیاز داریم که ماتریس را بچرخانیم تا بتوانیم درایه به درایه با ایندکس‌های برابر در هم ضرب کنیم به اصطلاح `transpose` می‌کنیم و در هر مرحله ۴ تا از درایه‌های ماتریس اول را در درایه‌های ماتریس دوم ضرب داخلی می‌کنیم و زمانی که تمام سطر در هم ضرب شد در اون ماتریس درایه حساب شده را قرار می‌دهیم. فایل این بخش نیز در `cross_parallel.asm` موجود است.

اگر برنامه‌ها را اجرا کنیم سرعت برنامه با استفاده از دستورهای موازی سریع‌تر است از برنامه با دستورهای عادی است و نشان می‌دهد که با استفاده از دستورهای موازی می‌توانیم دستورها را سریع‌تر کنیم و برنامه بهتری داشته باشیم.

من ۱۰ دفعه نمونه گیری کردم از اجرای این دو برنامه با ماتریس‌های ورودی ۳ در ۳ که نتایج زمان اجرای آن‌ها را می‌توانید در جدول زیر ببینید :

	1	2	3	4	5	6	7	8	9	10	average
parallel	0.375	0.378	0.392	0.412	0.389	0.389	0.386	0.391	0.377	0.386	0.387
normal	0.453	0.405	0.385	0.392	0.418	0.443	0.387	0.432	0.384	0.424	0.413

همانطور که انتظار میرفت سرعت اجرای برنامه با دستورات موازی بیشتر است اما چون نمونه ما کوچک است تغییر به اندازه کافی بزرگ و قابل درک نیست.

حالا با استفاده از یک حلقه نیز در برنامه قسمت ضرب هر برنامه (نه قسمت ورودی و خروجی دادن) دو ماتریس ۳ در ۳ را یک میلیون بار محاسبه کردم و خروجی زمان آن در شکل‌های پایین آمده :

```
kasrahmi@hmi:/media/share/x86_template$ head -2 cross_parallel_output.txt
Time calculating parallel cross in ms :
41
```

شکل ۱ – زمان اجرای مربوط به برنامه موازی

```
kasrahmi@hmi:/media/share/x86_template$ head -2 cross_normal_output.txt
Time calculating normal cross in ms :
78
```

شکل ۲ – زمان اجرای مربوط به برنامه عادی

تصویرهای پایین نیز اجرای ضرب دو ماتریس ۱۰۰۰ در ۱۰۰۰ با استفاده از دستورات موازی و عادی در زبان اسکرپتی است :

```
time ./normal.sh < testCase.txt
real      0m3.687s
user      0m3.542s
sys       0m0.095s
```

شکل ۳ – استفاده از دستورات عادی برای ضرب ماتریس

```
time ./run.sh < testCase.txt

real    0m1.365s
user    0m1.202s
sys     0m0.114s
```

شکل ۴ – استفاده از دستورات موازی برای ضرب ماتریس

تصویر بالا مربوط به دستورات عادی است که سرعت کمتری دارند اما تصویر پایین مربوط به برنامه‌ای است که در آن از دستورات موازی استفاده کردیم و همانگونه که انتظار می‌رفت سرعت برنامه‌ای که از دستورات موازی استفاده می‌کند نزدیک به ۳ برابر بیشتر است.

در مقایسه با برنامه‌ها با زبان سطح بالا نیز سرعت بیشتری دارند چون محاسبات در سطح پایین‌تری انجام می‌شود و می‌توانیم نتیجه بگیریم میتوانیم از زبان اسmbلی کمک بگیریم تا سرعت اجرای برنامه را افزایش دهیم به این خاطر که از زبان‌های سطح بالا سریع‌تر هستند و تجربه بهتری کسب کنیم.

```
● kasrahmi@amirkasras-MacBook-Pro-2: x86_template % time python3 2.py
Cross product written to 'cross_product.txt'.
python3 2.py 14.68s user 0.09s system 99% cpu 14.854 total
```

شکل ۵ – استفاده از کد پایتون برای ضرب ماتریس

تصویر بالا مربوط به زمان اجرای یک برنامه به زبان پایتون که زبان سطح بالا است برای ضرب ماتریس است که نشان می‌دهد حدود ۱۵ برابر کنتر از برنامه با زبان اسmbلی ما است پس می‌توانیم نتیجه بگیریم استفاده از زبان اسmbلی باعث افزایش سرعت و کارایی نسبت به زبان‌های سطح بالا شود.

بخش دوم سوال :

برنامه خود را برای محاسبه یک تابع 2D-Convolution دلخواه (مثلاً برای یک کار پردازش تصویر یا هوش مصنوعی) به کار ببرید و زمان اجرای برنامه کامل را با برنامه‌های مشابه موجود یا خودتان مقایسه کنید. توضیح اینکه عمل Convolution (به زبان ساده) یک تابع را بروی محور افقی از روی یک تابع دیگر عبور می‌دهد و در هر نقطه برخورد (یا در نقاط گسته) حاصل ضرب این دو تابع را محاسبه و ثبت می‌کند. (تعداد قابل توجهی از پردازش‌های تصویر و اخیراً مذل‌های یادگیری ماشین، بر اساس تابع Convolution پیاده‌سازی شده است). میتوانید یک تابع را با ماتریس 5 در 5 و دیگری را 3 در 3 در نظر بگیرید.

دو برنامه متفاوت برای اجرای کانولوشن طراحی کردم یکی با ماتریس (کرنل) 3 در 3 و یکی 5 در 5 برای طراحی این برنامه ابتدا ماتریس‌های خودم را تبدیل به ماتریس‌های بزرگتری کردم که مانند [این منبع](#) بتوانم برنامه کانولوشن خود را پیاده سازی کنم، در برنامه‌ای که کرنل 3 در 3 دارد یک ردیف و یک ستون از هر طرف اضافه می‌شود و در برنامه‌ای که کرنل 5 در 5 دارد به هر طرف باید دو ردیف و ستون اضافه کنیم تا بتوانیم از ماتریس جدید استفاده کنیم.

زمانی که سرعت اجرای این برنامه را با برنامه‌های آماده با زبان‌های سطح بالا بررسی کردم متوجه شدم که در اینجا نیز سرعت اجرا برنامه اسمنلی بیشتر است و دوباره برتری این پروژه را متوجه شدیم.

برای اجرای برنامه اسمنلی خود نیاز است ماتریس روشنایی هر پیکسل را به آن بدهم تا با ضرب آن ماتریس در یک کرنل (با توجه به خواسته خود) ماتریس کانولوشن را بسازم و آن را تبدیل به عکس کنم، پس از دو برنامه پایتون نیز برای تبدیل عکس به ماتریس روشنایی و تبدیل ماتریس روشنایی به عکس استفاده می‌کنم.

در برنامه زمان اجرای کانولوشن را در زبان اسmbلی و کد پایتون مشابه آن را محاسبه کرده‌ام که می‌توان نتیجه زمان اجرای آن‌ها را در تصاویر پایین ببینید :

```
kasrahmi@hmi:/media/share/x86_template$ head -2 convolution_output.txt
Time calculating convolution in ms :
266
```

شکل ۶ – زمان اجرای کانولوشن در برنامه اسmbلی

```
● kasrahmi@amirkasras-MacBook-Pro-2 x86_template % time python3 3.py
python3 3.py 1.17s user 0.04s system 96% cpu 1.266 total
```

شکل ۷ – زمان اجرای کانولوشن در برنامه پایتون

همانطور که می‌توانید مشاهده کنید زمان اجرای برنامه اسmbلی حدود ۵ الی ۶ برابر سریع‌تر است و در اینجا هم می‌بینیم که کارایی و سرعت در برنامه‌ای که به زبان اسmbلی نوشته شده بسیار بیشتر و تاثیرگذار‌تر از زبان‌های سطح بالا مانند پایتون است.

در صفحه بعد می‌توانید نمونه‌هایی از عکس‌های تولید شده توسط برنامه کانولوشن اسmbلی که در این پروژه آماده شده است را مشاهده کنید.

تست‌هایی از اجرای برنامه کانولوشن:

• عکس اصلی :



شکل ۸ - تصویر اولیه

عکس تولید شده با کرنل $\begin{matrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{matrix}$ که برای وضوح تصویر لکه دار مفید است و به اصطلاح فیلتر sharpen را اعمال می‌کند:

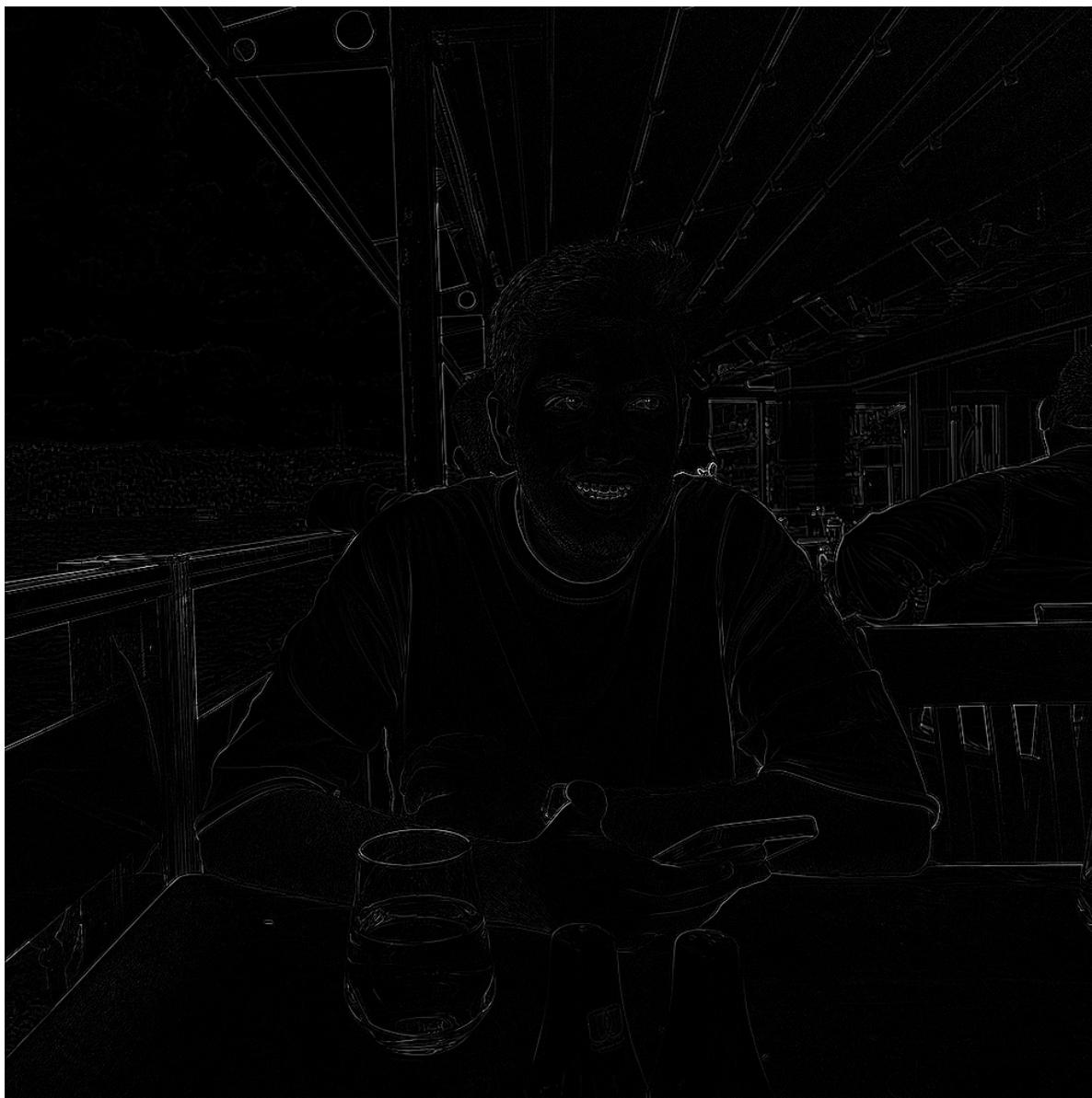


شکل ۹ – تصویر تولید شده با کرنلی که برای وضوح بیشتر است

عکس تولید شده با کرنل ridge که برای برآمدگی مرزها است و به اصطلاح فیلتر

$$\begin{matrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{matrix}$$

را اعمال می‌کند:



شکل ۱۰ – تصویر تولید شده با کرنلی که برای نمایش مرزها است

$$\begin{matrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{matrix}$$

عکس تولید شده با کرنل blur را اعمال

می‌کند :



شکل ۱۱ – تصویر تولید شده با کرنلی که برای تار کردن است

در نهایت نیز با توجه به تجربیاتی که در طول پروژه کسب شد می‌توان نتیجه گرفت که استفاده از زبان اسمنبلی چه مقدار می‌تواند به سرعت برنامه‌های ما کمک کند و برنامه سریع‌تری نسبت به برنامه با زبان‌های سطح بالا به ما تحويل دهد ، حتی در همان برنامه‌های اسمنبلی نیز استفاده از دستورات موازی سازی نیز می‌تواند سرعت اجرای برنامه را سریع‌تر نیز بکند و باعث بهبود چندرابری در زمان اجرای برنامه‌ها شود.

منابع :

- https://en.wikipedia.org/wiki/Convolution#/media/File:Convolution_of_spiky_function_with_box2.gif
- [https://en.wikipedia.org/wiki/Kernel_\(image_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing))
- https://en.wikipedia.org/wiki/Convolution_theorem)