



**T.C.
GEBZE TEKNİK ÜNİVERSİTESİ**

Bilgisayar Mühendisliği Bölümü

**SYSTEM PROGRAMING
MIDTERM**

**FÜRKAN YILDIZ
141044031**

Convolution

Convolution sinyal ve görüntü işleme operasyonlarında kullanılır. 1 boyutlu convolution, 2 signal üzerinde, 2 boyutlu convolution ise 2 görüntü üzerinde çalışır. Bu 2 liklerden birini input sinyali/görüntüsü, diğerini kernel/ input görüntüsü için filtre olarak düşünebiliriz. Bu operasyon çıktı olarak ise 3. bir sinyal ya da görüntü üretir.

1 boyutlu convolution, bir boyutlu görüntü ya da bir boyutlu sinyal, bir boyutlu vektör ile temsil edilir.

2 boyutlu convolution ise matrisler ile temsil edilir. Bu yöntem, görüntülerin pürüzsüzleştirilmesi, keskinleştirilmesi ve kenar algılanması gibi görüntü işlemede sıklıkla kullanılan bir yöntemdir.

Alt yapıya baktığımızda ise matrislerin elemanlarının kernel matris ile çarpılıp toplanmasından başka bir şey yoktur. Bu çarpım yapılırken ilk önce input matrisin en üst sol elemanından başlanır ve kernel ile çarpılıp toplandıktan sonra sırasıyla input matris gezilir, aynı işlemler devam ettiğinde ise output matris (görüntü) elde edilir.

$$(f * g)(i) = \sum_{j=1}^m g(j) \cdot f(i - j + m = 2)$$

f: Input vector

g: Kernel

n: input'un boyutu

m: Kernel'in boyutu

1	2	1	
0	0	1	2
-1	-2	4	5
	7	8	9

$$\begin{aligned} y[0,0] &= x[-1,-1] \cdot h[1,1] + x[0,-1] \cdot h[0,1] + x[1,-1] \cdot h[-1,1] \\ &\quad + x[-1,0] \cdot h[1,0] + x[0,0] \cdot h[0,0] + x[1,0] \cdot h[-1,0] \\ &\quad + x[-1,1] \cdot h[1,-1] + x[0,1] \cdot h[0,-1] + x[1,1] \cdot h[-1,-1] \\ &= 0 \cdot 1 + 0 \cdot 2 + 0 \cdot 1 + 0 \cdot 0 + 1 \cdot 0 + 2 \cdot 0 + 0 \cdot (-1) + 4 \cdot (-2) + 5 \cdot (-1) = -13 \end{aligned}$$

... (şeklinde devam edildikten sonra)

			m			
		n		-1	0	1
		-1	-1	-2	-1	
		0	0	0	0	
		1	1	2	1	
1	2	3				
4	5	6				
7	8	9				

Input matrix

Kernel matrix

Output matrix

Aşağıdaki resimlerde 2d convolution'un resimler üzerinde yaptığı değişiklikler örneklenmiştir. İlk örnekte resimdeki her piksel identity matris ile çarpılmış ve ana resmin aynısı output olarak elde edilmiştir. Diğer resimlerde ise farklı matrisler ile bu çarpma işlemleri yapılarak farklı outputta resimler elde edilmiştir.



	*	<table> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>1</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td></tr> </table>	0	0	0	0	1	0	0	0	0	=	
0	0	0											
0	1	0											
0	0	0											
Original					Identical image								

	*	$\frac{1}{9}$ <table> <tr><td>1</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </table>	1	1	1	1	1	1	1	1	1	=	
1	1	1											
1	1	1											
1	1	1											
Original					Blur (with a mean filter)								

	*	<table> <tr><td>0</td><td>0</td><td>0</td></tr> <tr><td>1</td><td>0</td><td>0</td></tr> <tr><td>0</td><td>0</td><td>0</td></tr> </table>	0	0	0	1	0	0	0	0	0	=	
0	0	0											
1	0	0											
0	0	0											
Original					Shifted left By 1 pixel								

Bu projede 3 adet program yazılması istendi. İlk program,

timeServer <ticks in msec> <n> <mainpipeName>

şeklinde çalışacak olan time server programı, bu program, kendisine parametre olarak gelen

milisaniye kadar, kendisine gelen sinyalleri bloklayıp, uyuyacak, ardından uyandığında sinyalleri unblock edecek ve gelen sinyaller ile ilgilenecek, her sinyal için bir matris üretilcek. Üretilen bu matris, kendisine girilen 2. Parametre olan n'in iki katı kare bir matris üretecek yani $2n \times 2n$ boyutunda. Üretilen bu matrisin tersi alınabilir olacağından, $\text{rand}(0 - 1000) \pm 500$ aralığında matris üretiliyor. Bu matrisin determinantı kontrol ediliyor ve eğer determinant sıfır ise bu matris tekrar üretiliyor. Böylece elde edilen matrisin tersinin alınabileceği garantileniyor. Üretilen bu matrisin see What programına gitmesi için bir fifo dosyası açacağız fakat bu fifo dosyasının adını hem server hemde clientin bilmesi için bu fifo dosyasına isim olarak clientin pid'si veriliyor. Clientin pid'sini server'a bildirebilmek için ise, clientten "SIGUSR2" sinyali gönderiliyor. Server bu sinyali aldığı anda içerisinde bulundurduğu client arrayine, bu pid'i ekliyor. Böylece 2 tarafta matrisin yazılıp okunacak dosyayı biliyor. Server'ın içerisinde client arrayi tutulmasının nedeni ise, clientlerin birden fazla olabileceğidir. Bu şekilde server tüm clientlerin pid'sini bilir ve işlemlerini ona göre yapar. Clientin, servera sinyal yollayabilmesi için ise onun pid'sini bilmesi gerekir. Bu durum ise serverın pid'si "mainpipeName" dosyasına yazılıp, client tarafından okutularak çözülmektedir. Matrisin üretilmesi ise seeWhat'an gelen isteğe göre gerçekleşiyor. seeWhat matris istediğinde, server'a "SIGUSR1" sinyalini gönderiyor ve server sinyali aldığı anda matris üretme işlemleri başlatılıyor. Sinyal alındığında matrisi üretmek için bir fork işlemi gerçekleştiriliyor ve matris çocuğa üretiliyor, bu sırada main process ise işlemlere devam ediyor (başka sinyal, geldimi, geldiyse matris üreteyim) böylelikle paralellik sağlanıyor ve matrisler teker teker değil aynı anda üretilabiliyor. Matris üretildikten sonra ise, matrisin üretilme zamanı ms olarak (server çalışmaya başladığından itibaren, ilgili matrisin oluştuğu milisaniye, program çalışınca ve matris oluşturulunca "gettimeofday" fonksiyonu çağırılıyor ve birbirinden çıkartılıyor.), elde edilen matrisin determinantı ve matrisin üretilmesi için sinyal gönderen clientin pid'si server'ın log dosyasına yazılıyor. Sinyali gönderenin, pid'si ise "sigaction" fonksiyonu ile elde ediliyor. Bu fonksiyon sinyali gönderen ile ilgili bir struct tutuyor ve bu struct gönderenin pid'sinde sahip. Bundan faydalanılıyor Clientin pid'sini elde etmek için.) Daha sonra seeWhat ile iletişim kurmak üzere açılan fifo dosyasına, önce matrisin boyutu sonra ise matrisin elemanları tek tek yazılıyor ve fifo kapatılıyor. Child process'in bu işlemlerden sonra artık bir görevi kalmadığından child process öldürülüyor. Main process ise bu işlemler gerçekleşirken sürekli "ticks in msec" sürece sinyal gelip gelmediğini kontrol ediyor. Bu işlemler program terminate ettirilene kadar gerçekleşeceğinden işlemler "while(1)" döngüsünün içerisinde gerçekleşiyor. Server'ın içerisinde program terminate ettirilir ise yani "CTRL + C" ye basılır ise tüm aktif olan terminallerin kapatılması gerekli. Ayrıca terminate ettirilen zamanda server'ın log dosyasına yazdırılacak. Bunu sağlamak için "SIGINT" sinyalini handle etmemiz gerekli. Bunun için sigactina, "SIGINT" i ekliyoruz. "SIGINT" sinyali geldiğinde ise sinyalin geldiği zamanı (time -localtime ve asctime fonksiyonlarını kullanarak) dosyaya "Şu saatte CTRL + C" geldi şeklinde yazıyoruz. Daha sonra ise, client oluştuğunda, servera sinyal gönderterek oluşturduğumuz client arrayi üzerinde gezerek, tüm clientlere onları öldürmek için "SIGUSR2" sinyali gönderiyoruz. Daha sonra clientlerin kendi pid'leri ile oluşturdukları fifo dosyalarını (client arrayi tuttuğumuz için bunların ne olduğunu biliyoruz) ve "mainpipeName" fifo dosyasını siliyoruz. Böylece ortada kalıntı bırakmamış oluyoruz. Dışarıdan kill sinyali geldiğinde ise (Diğer terminallerden birince "CTRL + C" yapıldığında) server'a "SIGALRM" sinyali gönderiliyor. Bu sinyali alan server, log dosyasına sinyalin geldiği zamanı (time -localtime ve asctime fonksiyonlarını kullanarak) yazıyor ve SIGINT sinyali geldiği zaman yapıtımız gibi, clientlere, "SIGUSR2" sinyali gönderiliyor ve clientlerin oluşturdukları fifo ve main fifo dosyaları siliniyor. Böylece ortalıkta kalıntı kalmıyor.

seeWhat <mainpipename>

seeWhat programı, önce “mainpipename” e yazılmış olan serverin pid’sini okuyacak ve ardından servera kendini tanıtmak üzere (Server’a pid’sini öğretmek için(server’da clientlerin arrayi yer alıyor)) servera “SIGUSR2” sinyali gönderiyor. Kendini tanıttıktan sonra ise servera matris oluştur komutunu vermek için “SIGUSR1” sinyali gönderiliyor. Sinyali alan server matrisi oluşturup oluşturmaz, clientin pidsinin adıyla oluşturulmuş fifo dosyasına matrisi yazdıktan sonra fifo okunuyor (fifo dosyasının clientte açılması bir while dongusunun içerisinde çünkü server daha matrisi yazmadan client okumaya çalışıp, sonsuza dek bekleyebilir.) fifo okunurken önce matrisin boyutu, daha sonra ise matrisin elemanları okunuyor, okunan boyuta göre kare matris olacağından matrisimiz dolduruluyor matris. Okunan bu matrisi seeWhat programının log dosyasına yazmamız gerekmekte. Programın log dosyalarını, diğer clientler çalıştığında onların log dosyaları ile karışmaması için “seeWhat_PIDOFCLIENT_NUMOFMATRIX” formatında yazıldı. Böylece birden fazla client çalıştırıldığında üst üste yazma sorunu çözüldü. Açılan clientin log dosyasına önce okunan matris direk “orijinal = [...]” formatında yazılıyor ve dosya kapatılıyor. showResut programı çalıştığında result1,result2,timeElaps1,timeElaps2 ve client’in pid’sine ihtiyaç duyacağı için showResut programına bunları iletmek adına, client ve showResult programları arasında bir fifo dosyası oluşturacağız, bu fifo dosyasıyla bu değerleri showResult’a göndermeliyiz, bunun için client’e bu değişkenleri içeren(result1,result2,timeElaps1,timeElaps2 ve client’in pid’si) bir struct oluşturdum. Ardından Inverse ve 2dConvolution için iki adet matrix oluşturuluyor, bu matrisler orijinal matrisin elemanları ile dolduruluyor ve hemen ardından fork işlemi gerçekleştiriliyor. Oluşan child process, inverse işlemini yapacak, bunun için “inverse” fonksiyonunu çağırıyor (Inverse fonksiyonu 4 adet sub matrix üretiyor (Ana matrisin size’ının yarısı boyutunda) ve ana matrisimizi 4’e bölerek her matrisin tek tek tersini almak için ters alma işlemini gerçekleştiren fonksiyonu çağırıyoruz böylece elimizde 4 matrisin tersi alınmış şekilleri oldu bu işlem ardından ise an matrisi böldüğümüz formatta bu 4 matrisi birleştiriyoruz.) Daha sonra Inversi alınmış matrix seeWhat’ın log dosyasına “Inversed = [...]” formatıyla yazılıyor. Bu işlemden sonra result1 ve timeElaps oluşturmamız lazım. Result1 = det(main matrix) – det (inversed matrix), timeElaps ise result1 i hesaplarken geçen süre. Bu iki değeri hesapladıktan sonra bir pipe oluşturuyoruz ve pipe’a bu iki değeri yazıyoruz. Main process ise pipe’dan bu değerleri, struct’ımızın result1 ve timeElaps1 değerlerine okuyor. Ardından child process ölüyor ve main process 2d convolutionu hesaplamak için yeni bir fork yapıyor. Oluşan child process “doConvolution” fonksiyonunu çağırıyor ve bu fonksiyonda gelen matrix 4 e bölünerek her parçası için ayrı ayrı convolution fonksiyonu çağırılıyor ve her parça aynı şekilde yerine yazılıyor. Bu kısımda kullandığımız kernel matrix identity olduğu için orijinal matris ile aynı matris oluşuyor convolution sonucu. Convolution matrisimizde oluşturduktan sonra seeWhat’ın log dosyasına “2dConverted = [...]” formatında yazıyoruz. Daha sonra result2’yi hesaplıyoruz, 2 determinant fonksiyonu çağırısı ile result2 = det(ana matris) – det(2d convoluted matris) ve result2 oluşurken geçen süreyi hesaplıyoruz öncesine ve sonrasına timer koyup birbirinden çıkartarak. Ardından bu değerleri pipe’a yazıyoruz. Main process bu değerleri struct’un result2, timeElapsed2 değişkenlerine okuyor. Child process i kapatıyoruz ve ardından struct’a client’in pid’sinide “getpid()” ile doldurduktan sonra struct’u showResult’un okuması için fifo dosyasına yazıyoruz. Bu ffonun adı ise “PIDOFCLIENT_RESULTS”. Clientte “CTRL + C” komutu gelirse, “SIGINT” sinyali handle ediliyor. Log dosyasına sinyalin geldiği zamanı ve “CTRL + C” ya basıldığını yazıyor. Ardından server’e ve showResult programlarına kendilerini terminate etmeleri için “SIGALRM” sinyali gönderiliyor. Bu sinyali showResult’a gönderebilmek için ise showResult’un pid’sini bilmesi gerekmekte bunu öğrenebilmek için showResult, client’e “SIGUSR1” sinyali gönderiyor ve client bunu store ediyor. Server yada showResult programları terminate ettiğinde ise, seeWhat programına “SIGUSR2” sinyali gönderiliyor, bu sinyalin alındığı saati ve kill sinyalinin geldiğini log dosyasına yazılarak program sonlanıyor.

showResults

Bu program çalıştırıldığı andan itibaren aktif processlerin oluşturdukları matrislerin result1 result 2 ve pidlerini ekrana, bunlarla beraber resultların hesaplama sürelerininide log dosyasına yazacak. Bu program çalışmaya başlayınca 2 adet boş liste oluşturuyor bu listelerden birisi üzerinde işlem yapılan clientlerin pid'lerini tutarken diğeri timeSever'ın log dosyasında gezerek farklı pid'leri tutan bir liste.

Program timeserver'ın log dosyasından clientin pid'leri çekecek ve çektiği pid'lere göre client ile iletişime geçecek (içerisinde result1,retulst2, timeElaps1, timeElaps2 ve clientin pid'si yer alan fifo dosyasını okuyacak ilgili clientin) Bunu yaparken timeserver'ın log dosyası her an güncellenmesi için ürettiğim çözüm şu şekilde, program sonsuz kere çalışacağından while(1) içerisinde ve her seferinde yeni bir pid eklenmiş mi diye kontrol ediyor timeserver.log dosyasını. Yeni bir pid bulduğunda onun için bir fork oluşturuyor ve child process herhangi bir terminalden terminate komutu gelene kadar sürüyor, yani kısacası showResultta her client için bir process oluşturuluyor ve bu process programlardan biri terminate edene kadar devam ediyor. Böylece her client'in yeni matrislerini yakalıyor ve daha sonrada client açılrsa tekrar onuda yakalayabiliyor ve onun içinde bir process oluşturuluyor. Processin yaptığı iş ise clientlerin pid'sini aldıktan sonra "PIDOFCLIENT_RESULTS" adında clientte oluşturulan ve içerisinde result1,retulst2, timeElaps1, timeElaps2 ve clientin pid'si yer alan struct'u, yine aynı değişkenler ile oluşturulan showResult taki struct'a okuyor. Böylece elimizde sahip olmamız gereken tüm bilgiler yer alıyor. Bunları aldıktan sonra ekrana clientin pid'ini ,result1'i ve result2'yi yazıyor. showResult'un log dosyasına ise structtaki tüm bilgileri yazıyor.

Eğer bu programın içerisinde "CTRL + C" yapılırsa program kendi log dosyasına sinyalin geldiği zamanı ve sinyalin eldiğini yazacak, client ile oluşturduğu fifo dosyasını silecek ve ardından clientlere "SIGINT" sinyali gönderecek ve clientler "SIGINT" sinyalini aldığıında onlarda server'a sinyal gönderecek ve böylece 3 programda kapanacak.

Diger terminallerden birinde "CTRL + C" yapıldığında ise showResult'a "SIGALRM" sinyali gelecek ve log dosyasına ne zaman geldiğiyle beraber kill sinyali geldiğini basacak ardından client ile arasındaki fifoyu silerek programı sonlandıracak

CIKTIJAR AYRI OLARAK BELIRTILDI. DOSYAYA KOYULDU.

The image shows a Linux desktop environment with a purple-themed desktop background. In the foreground, there are two windows:

- Terminal Window:** Displays the execution of a C++ program. The program is located at `~/Desktop/Untitled Folder/141044031/Log/timeServer.log`. The output shows a list of timestamps and IP addresses, such as `882.395000.ms 47758 264097198478189792380059648.000000`.
- Sublime Text Window:** Displays the output of the program, showing a list of timestamps and IP addresses, such as `882.395000.ms 47758 264097198478189792380059648.000000`.

The Sublime Text window has multiple tabs open, including `seeWhat.c`, `seeWhat_47758_243.log`, `showResults.log`, `OTE`, `timeServer.c`, `showResults.c`, and `timeServer.log`. The `timeServer.log` tab is currently active, showing the output of the program.

The image shows a Linux desktop environment with a purple-themed desktop background. A dock on the left side contains icons for various applications, including a file manager, a web browser, a terminal, and a music player. The main window is a Sublime Text editor, which is open to a log file named 'Log/showResults.log'. The log file contains several lines of text, each representing a performance measurement for a specific PID (47758). The text is organized into columns, with the first column containing the PID, the second column containing a timestamp, and the third column containing an elapsed time. The log file is displayed in a dark theme, with the text in white. The Sublime Text editor's interface includes a menu bar at the top, a toolbar on the left, and a status bar at the bottom. The status bar shows the current line and column (Line 1, Column 1) and the tab size (4). The desktop environment also includes a top panel with system icons and a clock showing 23:41.







