



Nhóm 25: QT

Lập trình song song

***IMPLEMENTING AND  
OPTIMIZING THE  
FORWARD-PASS OF A  
CONVOLUTIONAL  
LAYER USING CUDA***



01

20120554

Nguyễn Minh Quân



02

20120587

Nguyễn Hoàng Thịnh



# Mục lục



## Giới thiệu ứng dụng

Giới thiệu về ứng dụng

01

## Cài đặt tuần tự

Mô tả về quá trình  
cài đặt tuần tự

02

## Cài đặt song song cơ bản

Mô tả về quá trình  
cài đặt song song cơ bản

03

## Cài đặt song song + Tối ưu

Mô tả về quá trình  
song song và  
tối ưu

04

## Tự đánh giá

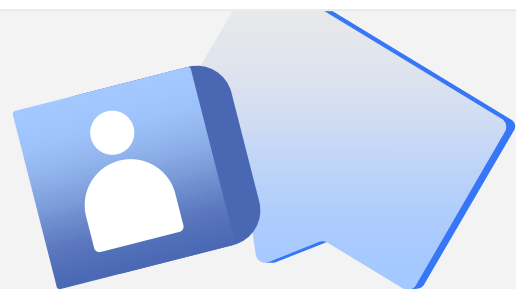
Tự đánh giá toàn bộ  
quá trình

05

## Tài liệu tham khảo

Các tài liệu  
đã tham khảo

06

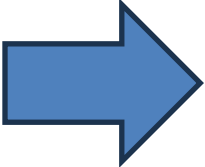




# 01. Giới thiệu ứng dụng

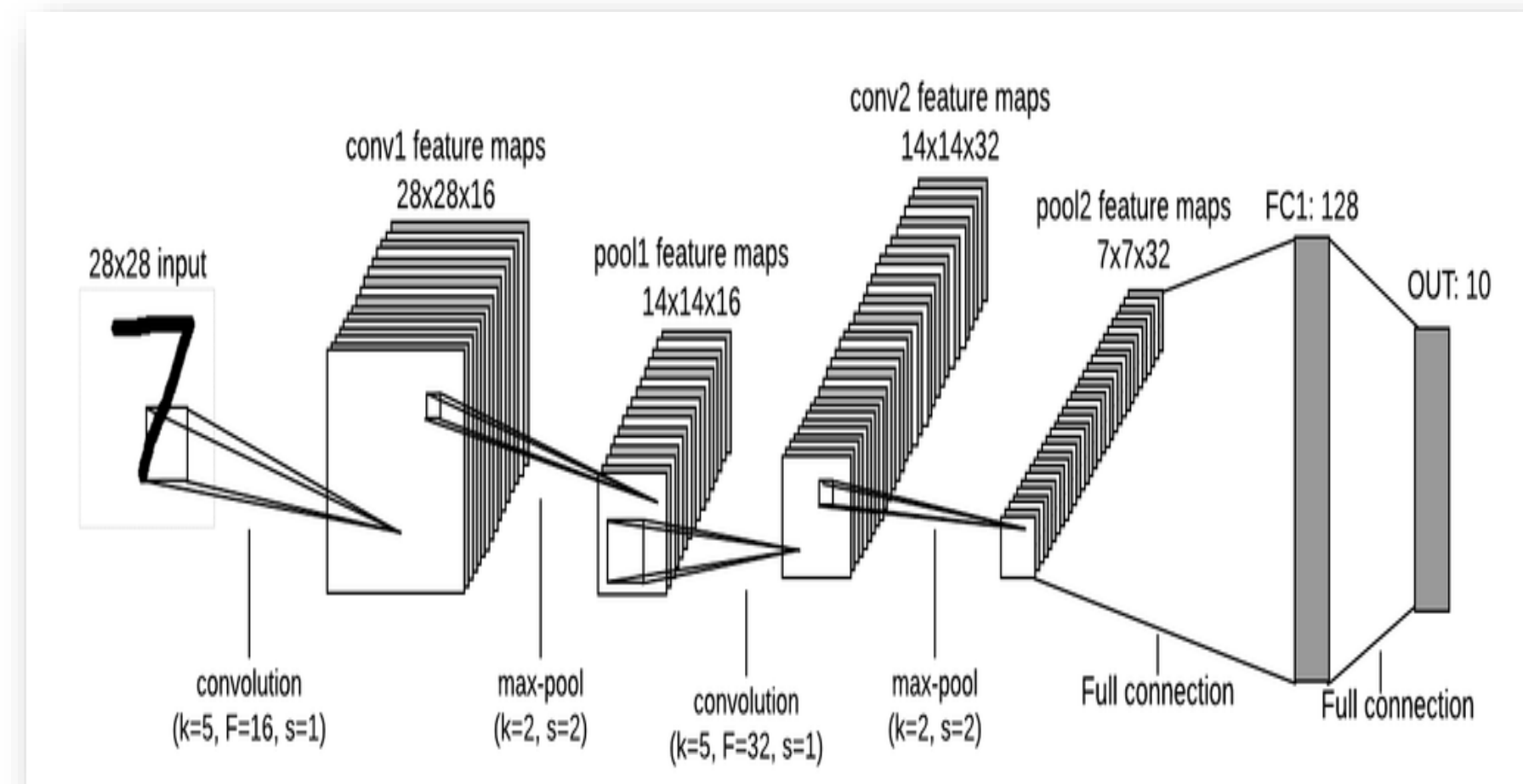
Giới thiệu về ứng dụng

# *Sơ lược ứng dụng*

 Tiến hành thực hiện và tối ưu giai đoạn forward-pass của lớp tích chập(Convolutional layer) bằng việc sử dụng CUDA để tiến hành song song hóa quá trình thực hiện lớp tích chập

# 1.1 Động lực thực hiện

- Các lớp tích chập là **các khối xây dựng chính** của mạng thần kinh tích chập (CNN), được sử dụng trong nhiều tác vụ học máy như phân loại hình ảnh, phát hiện đối tượng, xử lý ngôn ngữ tự nhiên và hệ thống đề xuất.
- Tuy nhiên, việc thực hiện lớp tích chập này khá tốn thời gian nên việc tối ưu hóa lớp tích chập là một điều cần thiết để tăng tốc độ của toàn bộ quá trình xây dựng của mạng thần kinh tích chập (CNN)

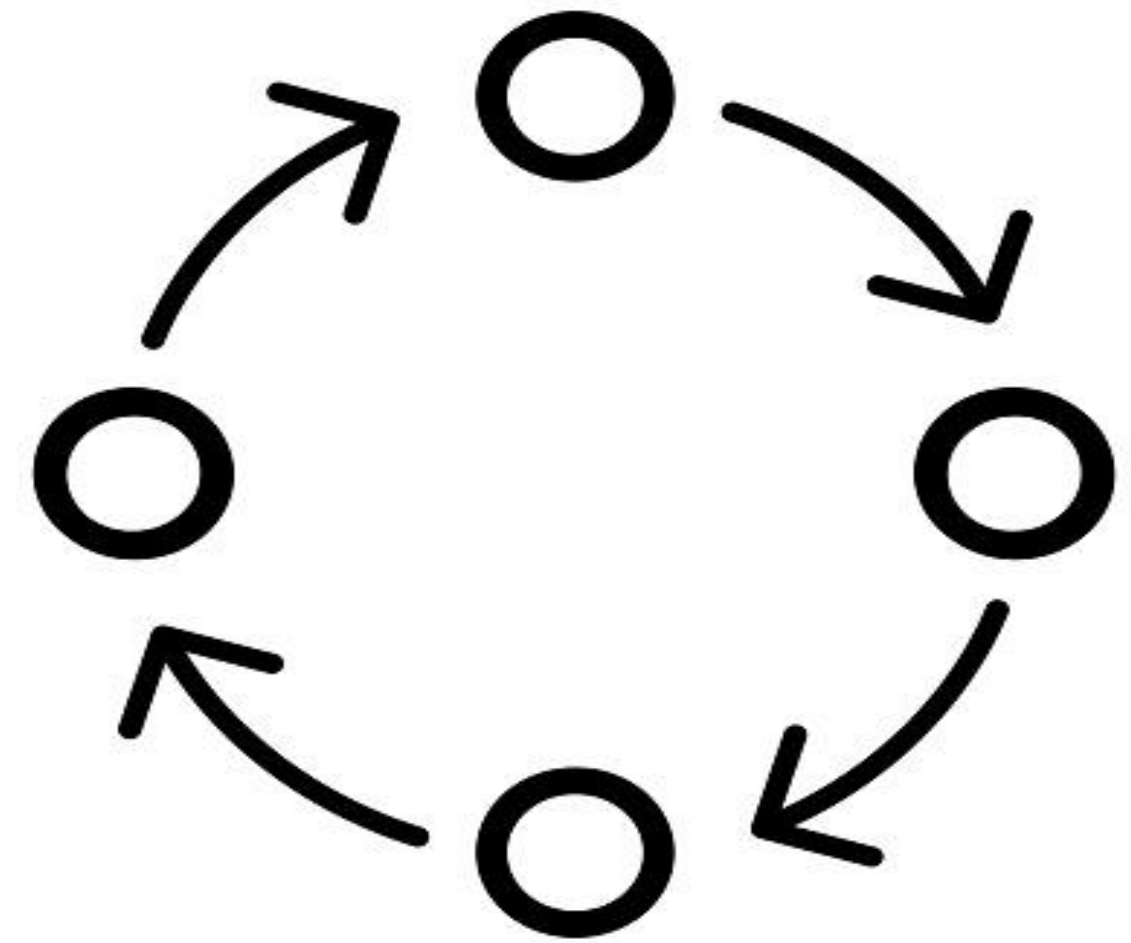


## 1.2 Tổng quan đóng góp

MSSV	Họ và tên	Công việc đã đóng góp	Phần trăm đóng góp
20120554	Nguyễn Minh Quân	Lên ý tưởng, thực hiện code phần song song cơ bản và các phiên bản tối ưu, viết document, viết report, làm video presentation	40%
20120587	Nguyễn Hoàng Thịnh	Thực hiện Makefile để biên dịch toàn bộ code, tìm hiểu, chạy code tuần tự, thực hiện code phần song song cơ bản và các phiên bản tối ưu, viết document, viết report, làm video presentation	60%

## 02. Cài đặt tuần tự

Mô tả về quá trình cài đặt tuần tự





## 2.1 Flow thiết kế

Tham khảo tại mini-dnn-cpp (Mini-DNN) framework

- ❑ Sử dụng một hàm `im2col`: Chuyển đổi một ma trận hình ảnh đầu vào định dạng cột ở dạng vector thành một ma trận dữ liệu `data_col` để chuẩn bị cho việc tích chập (`data_col` này có thể tích chập lại được với ma trận trọng số và bias)
- ❑ Thực hiện quá trình forward của lớp Convolution Neural Network (CNN) trong lớp tích chập với ma trận trọng số và bias cho các ma trận dữ liệu `data_col`.

## 2.2 Đánh giá

Tham khảo tại mini-dnn-cpp (Mini-DNN) framework

❑ Mô tả flow thực hiện:

❑ Ở hàm im2col:

1. Tiến hành duyệt qua từng điểm ảnh output (mỗi phần tử trong data\_col).
2. Với mỗi điểm ảnh output, nó tính toán vị trí bắt đầu của kernel trên ma trận hình ảnh input và lấy giá trị của các input tương ứng, hoặc thêm giá trị 0 nếu vị trí đó nằm ngoài biên của hình ảnh
3. Sau toàn bộ bước thực hiện trên thì thu được một ma trận dữ liệu data\_col

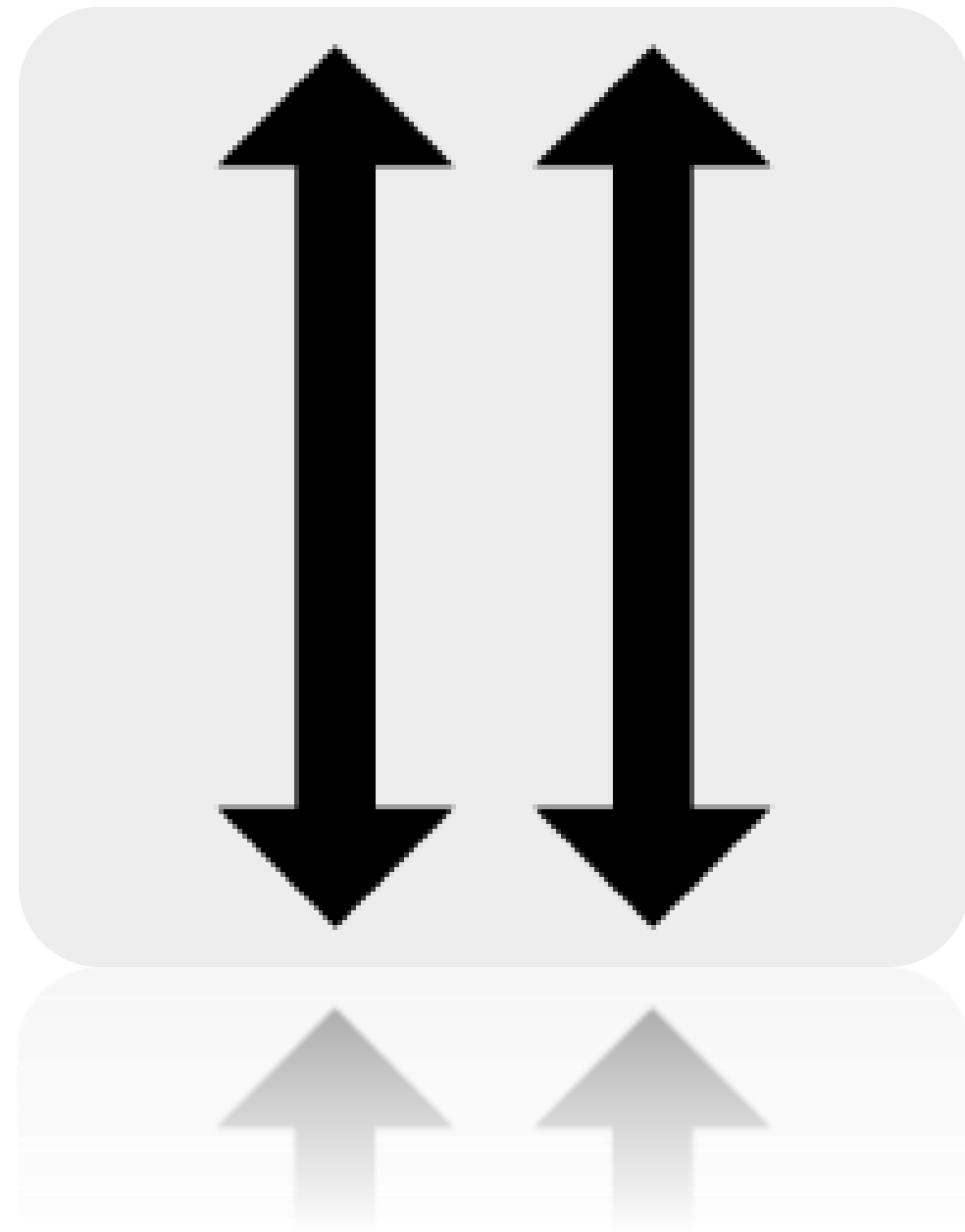
❑ Ở bước tính chập

1. Áp dụng việc tích chập với ma trận trọng số và bias cho từng mẫu trong dữ liệu input.

## 2.2 Đánh giá

- ❑ Tốc độ thực thi: Trung bình dao động từ 700 - 1300 ms (Tốc độ chiếm phần lớn toàn bộ trong quá trình thực hiện forward-pass)
- ❑ Tính đúng đắn: Chưa thực hiện
- ❑ Kết quả thực nghiệm: Chạy tốn nhiều thời gian

```
Idx: 0
Time for convolution layer by host (Sequential): 730.739 ms
Time for convolution layer by host (Sequential): 911.392 ms
Time for entire forward-pass: 1867.571 ms
Idx: 128
Time for convolution layer by host (Sequential): 734.445 ms
Time for convolution layer by host (Sequential): 931.927 ms
Time for entire forward-pass: 1875.280 ms
Idx: 256
Time for convolution layer by host (Sequential): 742.407 ms
Time for convolution layer by host (Sequential): 901.025 ms
Time for entire forward-pass: 1858.814 ms
Idx: 384
Time for convolution layer by host (Sequential): 744.593 ms
Time for convolution layer by host (Sequential): 1025.468 ms
Time for entire forward-pass: 2026.260 ms
Idx: 512
Time for convolution layer by host (Sequential): 1085.556 ms
Time for convolution layer by host (Sequential): 1342.186 ms
Time for entire forward-pass: 2722.234 ms
```



## 03. Cài đặt song song cơ bản

Mô tả về quá trình  
cài đặt song song cơ bản

# 03. Cài đặt song song cơ bản

## 3.1 Phân tích

Trong project này, chúng em song song hóa bước convolution. Vì bước này, những vị trí điểm ảnh output được tích chập không liên quan đến nhau nên mỗi vị trí điểm ảnh output sẽ được thực hiện ở một thread riêng.

## 3.2 Thiết kế

- Mục Tiêu: Tối ưu hóa việc thực hiện lớp tích chập bằng cách sử dụng song song hóa trên CUDA để gia tăng hiệu suất tính toán trong xử lý hình ảnh.
- Flow Thiết Kế: Thiết lập thuật toán tích chập song song trên CUDA. Sử dụng các khối tính toán song song hóa để tận dụng khả năng tính toán đồng thời trên GPU.
- Mô Tả Kỹ Thuật: Mỗi thread sẽ thực hiện tích chập trên một phần của input dựa vào mỗi bước nhảy của kernel và thread thứ  $i$  sẽ chứa kết quả của output thứ  $i$ .

# 03. Cài đặt song song cơ bản

## 3.3 Đánh giá

1. Thực hiện song song hóa hàm `im2col` để sinh ra ma trận `data_col` (Ma trận chưa được tích chập trọng số và bias)

Flow thực hiện:

- Cũng thực hiện tương tự như thực hiện tuần tự tuy nhiên thì hàm `im2col` được thực hiện song song bằng cách cho mỗi vị trí điểm ảnh output sẽ được thực hiện ở một thread riêng.

# 03. Cài đặt song song cơ bản

## 3.3 Đánh giá

❑ Tốc độ thực thi: Trung bình dao động từ 400

- 800 ms (Tốc độ chiếm phần lớn toàn bộ

trong quá trình thực hiện forward-pass

nhưng cũng đã giảm đáng kể so với tuần tự)

❑ Tính đúng đắn: Chưa thực hiện

❑ Kết quả thực nghiệm: Đã giảm thiểu được phân nửa thời gian chạy chương trình

Phương pháp 1:

```
Idx: 256
Time for convolution layer by kernel 0 (Parallel): 493.406 ms
Time for convolution layer by kernel 0 (Parallel): 746.242 ms
Time for entire forward-pass: 1435.232 ms
Idx: 384
Time for convolution layer by kernel 0 (Parallel): 488.886 ms
Time for convolution layer by kernel 0 (Parallel): 735.304 ms
Time for entire forward-pass: 1423.906 ms
Idx: 512
Time for convolution layer by kernel 0 (Parallel): 490.740 ms
Time for convolution layer by kernel 0 (Parallel): 734.366 ms
Time for entire forward-pass: 1421.529 ms
Idx: 640
Time for convolution layer by kernel 0 (Parallel): 495.913 ms
Time for convolution layer by kernel 0 (Parallel): 754.035 ms
Time for entire forward-pass: 1451.792 ms
Idx: 768
Time for convolution layer by kernel 0 (Parallel): 494.669 ms
Time for convolution layer by kernel 0 (Parallel): 740.796 ms
Time for entire forward-pass: 1450.607 ms
Idx: 896
Time for convolution layer by kernel 0 (Parallel): 492.553 ms
Time for convolution layer by kernel 0 (Parallel): 728.643 ms
Time for entire forward-pass: 1422.167 ms
```

# 03. Cài đặt song song cơ bản

## 3.3 Đánh giá

### 2. Thực hiện tích chập giúp đưa ra output đã tích chập với trọng số và bias (Song song)

Flow thực nghiệm:

- Xác định vị trí điểm ảnh output
- Với mỗi vị trí điểm ảnh output, dựa trên việc tính toán vị trí bắt đầu của kernel trên ma trận hình ảnh input và việc tính các vị trí điểm ảnh input tương ứng để thực hiện nhân với ma trận weight cộng dồn vào biến value.
- Sau khi cộng dồn hết vào value thì thực hiện công thêm với bias và sau đó trả kết quả cho output.



# 03. Cài đặt song song cơ bản

## 3.3 Đánh giá

- ❑ Tốc độ thực thi: Trung bình dao động từ 30 - 40 ms (Tốc độ đã giảm đáng kể so với toàn quá trình)
- ❑ Tính đúng đắn: Chưa thực thi
- ❑ Kết quả thực nghiệm: Đã tăng tốc chạy chương trình nhưng về tính đúng đắn thì chưa tốt

Phương pháp 2:

```
Idx: 128
Time Convolution: 35.594 ms
Time Convolution: 32.316 ms
Time for entire forward-pass: 285.586 ms
Idx: 256
Time Convolution: 35.752 ms
Time Convolution: 29.911 ms
Time for entire forward-pass: 285.998 ms
Idx: 384
Time Convolution: 36.434 ms
Time Convolution: 29.915 ms
Time for entire forward-pass: 286.330 ms
Idx: 512
Time Convolution: 39.249 ms
Time Convolution: 30.006 ms
Time for entire forward-pass: 308.702 ms
Idx: 640
Time Convolution: 37.822 ms
Time Convolution: 32.697 ms
```

# 04. Cài đặt song song + Tối ưu

Mô tả về quá trình song song và tối ưu



# 4.1 Tiled shared memory convolution

## 4.1.1 Phân tích

Quản lý shared memory hiệu quả có thể cải thiện hiệu suất. Nếu không quản lý shared memory tốt, sẽ dẫn đến xung đột và truy cập bộ nhớ không hiệu quả.

## 4.1.2 Thiết kế

Xác định kích thước block và kích thước tile để tối ưu hóa việc tận dụng shared memory và tài nguyên GPU. Mỗi thread block sẽ sao chép dữ liệu từ bộ nhớ toàn cục vào shared memory. Thực hiện tính toán convolution bằng cách sử dụng dữ liệu từ shared memory. Sau khi tính toán xong, ghi kết quả từ shared memory vào bộ nhớ toàn cục.

# 4.1 Tiled shared memory convolution

## 4.1.3 Đánh giá

Mô tả flow thực nghiệm:

- Xác định vị trí điểm ảnh output
- Tính toán kích thước tile trên shared memory
- Lưu input vào tile để có thể sử dụng được nhiều lần trên 1 block (\*)
- Dựa trên việc tính toán vị trí bắt đầu của kernel trên ma trận hình ảnh input và việc tính các vị trí điểm ảnh input tương ứng để thực hiện nhân với ma trận weight cộng dồn vào biến value. (\*)
- Sau khi cộng dồn hết vào value thì thực hiện công thêm với bias và sau đó trả ra kết quả cho điểm ảnh output

Lưu ý: Bước lưu và tính kết quả tạm thời đó phải syncthreads

# 4.1 Tiled shared memory convolution

## 4.1.3 Đánh giá

- ❑ Tốc độ thực thi: Trung bình dao động từ 30 - 40 ms (Tốc độ đã giảm đáng kể so với toàn quá trình)
- ❑ Tính đúng đắn: Chưa thực hiện
- ❑ Kết quả thực nghiệm: Đã tăng tốc chạy chương trình nhưng về tính đúng đắn thì chưa

```
Time Convolution: 43.530 ms
Time Convolution: 33.191 ms
Time for entire forward-pass: 292.272 ms
Idx: 128
Time Convolution: 46.481 ms
Time Convolution: 31.656 ms
Time for entire forward-pass: 293.771 ms
Idx: 256
Time Convolution: 39.885 ms
Time Convolution: 31.450 ms
Time for entire forward-pass: 284.860 ms
Idx: 384
Time Convolution: 41.352 ms
Time Convolution: 35.011 ms
Time for entire forward-pass: 314.053 ms
Idx: 512
Time Convolution: 40.663 ms
Time Convolution: 31.856 ms
Time for entire forward-pass: 294.338 ms
Idx: 640
Time Convolution: 40.221 ms
Time Convolution: 33.511 ms
Time for entire forward-pass: 288.690 ms
Idx: 768
Time Convolution: 40.741 ms
Time Convolution: 30.852 ms
Time for entire forward-pass: 289.730 ms
```

# 4.2 Weight matrix (kernel values) in constant memory

## 4.1.1 Phân tích

Constant memory được lưu trữ trên GPU và có thể cung cấp thời gian truy cập nhanh hơn. Việc sử dụng constant memory cho ma trận trọng số giúp tối ưu hoá truy cập dữ liệu.

## 4.1.2 Thiết kế

Chuẩn bị ma trận trọng số từ CPU và sao chép vào constant memory trên GPU. Sử dụng ma trận trọng số được lưu trữ trong constant memory trong quá trình tính toán convolution. Thực hiện tính toán convolution như bình thường, sử dụng ma trận trọng số từ constant memory.

# 4.2 Weight matrix (kernel values) in constant memory

## 4.1.3 Đánh giá

❑ Mô tả flow thực nghiệm:

- Sao chép ma trận trọng số từ CPU vào constant memory trên GPU trước khi chạy việc tích chập (Gồm khai báo biến constant và dùng `cudaMemcpyToSymbol` để copy từ host qua CMEM)
- Sau đó thì quá trình thực hiện như dùng tiled shared memory convolution

# 4.2 Weight matrix (kernel values) in constant memory

## 4.2.3 Đánh giá

```
Idx: 1024
Time Convolution: 37.944 ms
Time Convolution: 29.075 ms
Time for entire forward-pass: 285.468 ms
Idx: 1152
Time Convolution: 38.525 ms
Time Convolution: 31.028 ms
Time for entire forward-pass: 286.694 ms
Idx: 1280
Time Convolution: 34.953 ms
Time Convolution: 29.725 ms
Time for entire forward-pass: 293.291 ms
Idx: 1408
Time Convolution: 38.773 ms
Time Convolution: 29.439 ms
Time for entire forward-pass: 287.070 ms
Idx: 1536
Time Convolution: 35.531 ms
Time Convolution: 29.251 ms
Time for entire forward-pass: 285.595 ms
Idx: 1664
Time Convolution: 36.809 ms
Time Convolution: 30.732 ms
```

- ❑ Tốc độ thực thi: Trung bình dao động từ 30 - 40 ms (Tốc độ đã giảm đáng kể so với toàn quá trình)
- ❑ Tính đúng đắn: Chưa thực hiện
- ❑ Kết quả thực nghiệm: Đã tăng tốc chạy chương trình nhưng về tính đúng đắn thì chưa





## 05. Tự đánh giá

Tự đánh giá toàn bộ quá trình

## 5.1 Những khó khăn mà nhóm em gặp phải

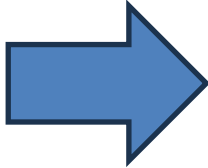
- ❑ Quân: Gặp khó khăn trong việc từ kernel ở lab 1 triển khai trong project này. Việc chạy code trên device thực sự khó khăn và tốn rất nhiều thời gian tìm hiểu (Chưa giải quyết được).
- ❑ Thịnh: Mất nhiều thời gian cho việc tìm hiểu các để có biên dịch code và chạy code trên device bằng Makefile (Đã giải quyết được). Khó khăn trong việc sử dụng thư viện Eigen cũng như là giải quyết các index của output, input, kernel

## 5.2 Những gì mà nhóm em học được?

- ❑ Quân: Em học được việc triển khai code trên project lớn. Sử dụng thư viện eigen và hiểu lớp tích chập thực hiện như thế nào.
- ❑ Thịnh: Học được các triển khai biên dịch nhiều file.cc và file.cu trên project lớn. Tìm hiểu được sơ lược thư viện Eigen, phép tích chập được thực hiện như thế nào khi thực hiện với tuần tự và song song.

## 5.3 Nếu có nhiều thời gian, nhóm em sẽ thực hiện

- ❑ Thực hiện code hoàn chỉnh toàn bộ phần cài đặt song song để đạt được tính đúng đắn,
- ❑ Tiến hành tính accuracy cho các cách cài đặt

 Từ đó, có thể xác định rõ hơn được về tính đúng đắn cũng như là kết quả thực nghiệm của cài đặt song song

## 06. Tài liệu tham khảo

- 01 mini-dnn-cpp:  
<https://github.com/iamhankai/mini-dnn-cpp>
- 02 Makefile tutorial  
<https://makefiletutorial.com/>
- 03 LeNet Architecture: A Complete Guide  
<https://www.kaggle.com/code/blurredmachine/lenet-architecture-a-complete-guide>
- 04 Fashion mnist:  
<https://github.com/zalandoresearch/fashion-mnist>

***Thanks for  
listening &  
watching!!!***

