Contents

1 This Week

1.1 extract feature

1.1.1 using caffe to extract features

1.1.2 general command for extract feature using caffe

- 1.caffemodel
- 2prototxt1caffemodellayer
- 3blobprototxtblob

blob

- 43

LMDB

- 5batchprototxtDataLayer

CaffeDataLayer(ImageDataLayer)$\text{batch}_{\text{size}}$ $\text{batch}_{\text{size}}$ * $\text{num}_{\text{minibatches}}$

- 6lmdbleveldb()

lmdblmdb

- 7GPUCPU

CPU

- 8GPUGPU

0GPU

- Test
  - DropoutTestdropout
  - TrainTestPrototxt (DataLayer)
- –
  –
- – Softmax(AlexNetfc8)

1.1.3 read from lmdb

1.1.4 image recognition using `cos` similarity measure

1.1.5 `cos` similarity result

- accuracy (true ture) : 53 / 55
- false true : 2 / 100

1.2 Cuda Note

1.2.1 Configuring the kernel launch

```
kernel«<grid of block, block of threads»>(...)
   square«<dim3(bx,by,bz), dime(tx,ty,tz), sharem»>(...)
```

- grid of blocks : bx * by * bz
- block of threads : tx * ty * tz
- shared memory per block in bytes

1.2.2 Convert color to black and white

$$I = (R + G + B)/3$$

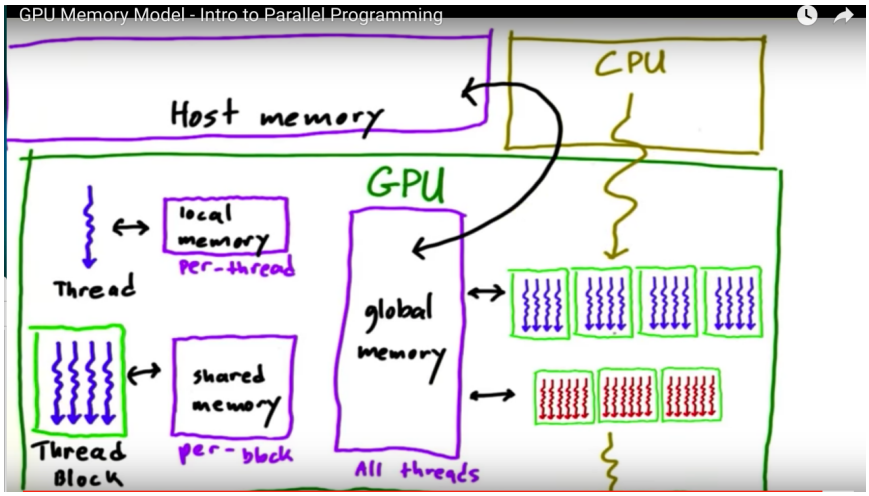
$$I = .299f * R + .587f * G + .114f * B$$

1.2.3 `nvcc` introduction

1.2.4 cs344 Note

- GPU is responsible for allocating blocks to SM(streaming multiprocessors)
- A block cannot run on more than one SM
- An SM may run more than one block
- All the SMs are running in parallel
- Threads in different block shouldn't cooperate
- Cuda make few guarantees about when and where thread blocks will run
- consequences
  - no assumptions blocks -> SM
  - no communication between blocks
- CUDA guarantees that:
  - all threads in a block run on the same SM at the same time
  - all blocks in a kernel finish before any blocks from next run
- threadIdx : thread within block threadIdx.x threadIdx.y
  - blockDim : size of block
  - blockIdx : block within grid
  - gridDim : size of grid

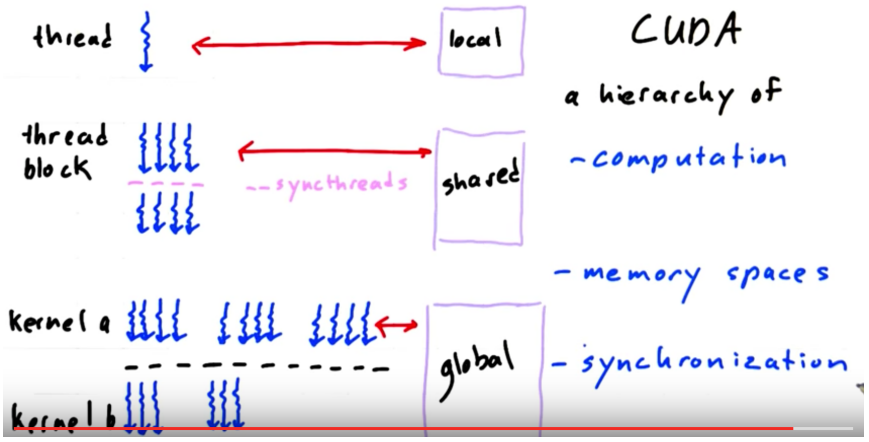
1.2.5 GPU memory model



- All threads from a block can access the same variable in that block shared memory
- Threads from two different blocks can access the same variable in global memory
- Threads from different blocks have their own copy of local variables in local memory
- Threads from the same block have their own copy of local variables in local memory

1.2.6 barrier

point in program where threads stop and wait. when all threads have reached the barrier, they can proceed.

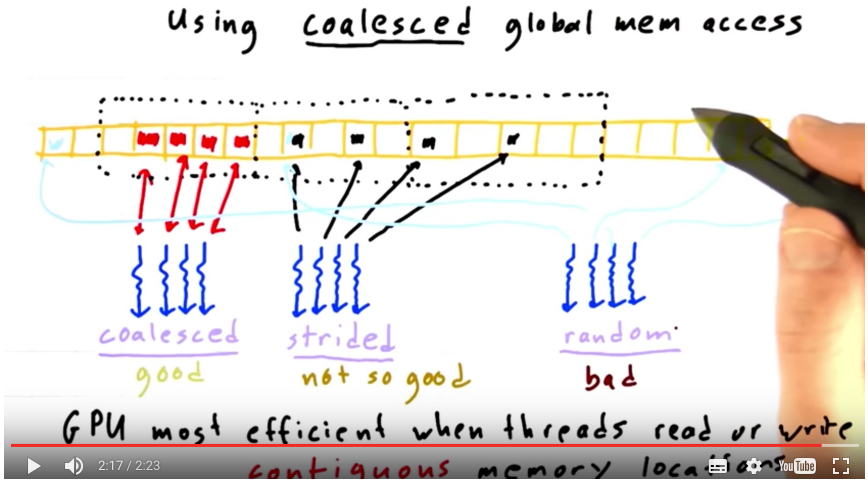


1.2.7 High-level strategies

1. Maximize arithmetic intensity

$$\frac{Math}{Memory}$$

- maximize compute ops per thread
- minimize time spent on memory per thread
  - move frequently-accessed data to fast memory local $>$ shared $\gg$ global $\gg$ cpu memory

coalesce memeory 

1. avoid thread divergence

1.2.8 `cudaMalloc`

`device_data device_data  cudaMalloc device_data`

1.2.9 TODO What Every Programmer Should Know About Memory

2 Next Week

2.1 Cuda programming

2.2 caffe