Digital Image Processing with GPU

Brice Ngnigha

Overview

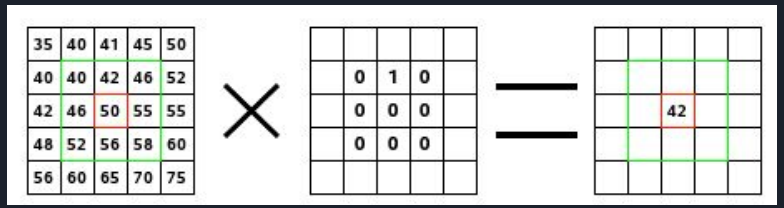
- Digital Image Processing is the use of computer algorithms to perform **image processing** on **digital** images.
- Major graphic applications such as such as photography, TV broadcasting use these algorithms to render.
- The major applications are in filtering noise, computer vision.
- We will discuss
  - Posing the Problem
  - Digital Filters
  - Design Constraints
  - The Memory Alignment and Management
  - Algorithm
  - Dynamic parallelism

The Problem to Solve

- Given a digital picture of size N by M
- We would like to apply some algorithms to introduce or extract some of the important features before moving it for further processing.
- We would like to solve this problem as fast as possible.
- We will use filters to extract the interesting features

Digital FIlters

- Filters form a subset of the techniques used to process digital images
- Many filters can be expressed via their mathematical form of convolution
- A convolution is a mathematical technique of combining two signals to form a third one
- Convolutions can be expressed as Matrices,. In the case of digital processing the multiplier matrix is called the **kernel matrix.**
- A signal can be obtained via this operation



(40*0)+(42*1)+(46*0) + (46*0)+(50*0)+(55*0) + (52*0)+(56*0)+(58*0) = 42

Digital FIlters

- The mathematical expansion of the Convolution of two matrices makes it a good candidate for parallel computation.
- Filtering a digital image now becomes a task of finding the right convolution kernel and apply it to the image
- Some kernels

Edge detection kernel

-1	-1	-1
-1	8	-1
-1	-1	-1

Laplacian of Gaussian

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

Designs Goals

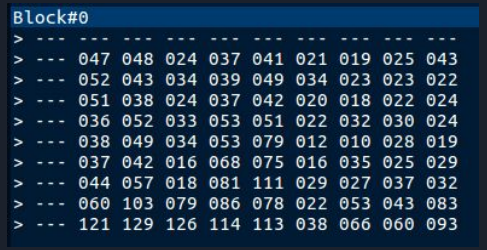
- Use parallelism to maximize Execution Time
  - Use the GPU + parallel algorithm
- Optimize Memory Access
  - Align memory for fast access
  - Manage Global memory access
- Reduce Thread Synchronization
  - Partition the data such that thread need not to always synchronize
- Optimize Work
  - Dedicate a single worker for shared computation

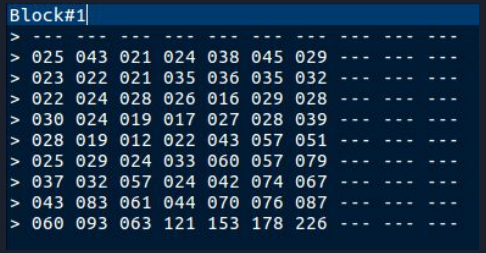
Memory Management and Alignment

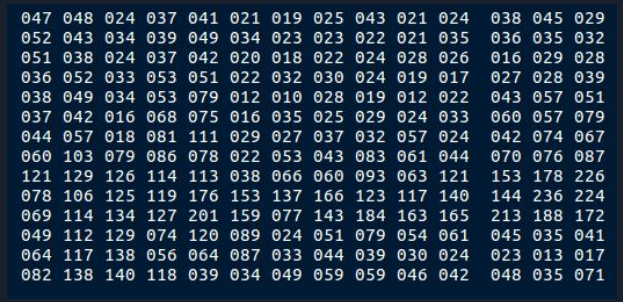
- Global Memory access is very expensive. We will copy all the shared memory needed within a block to avoid reading always from memory.
- Cache misses cre very expensive
- Align memory facilitate local access. Given a picture 14X14 with the below R(of RGBA) values. Memory is aligned in a 1 dimension array, row wise.

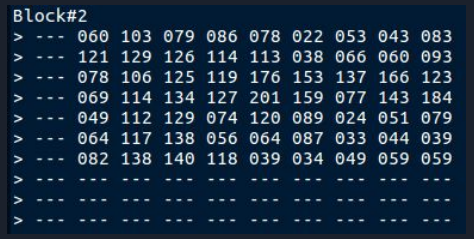
```
047 048 024 037 041 021 019 025 043 021 024  038 045 029
052 043 034 039 049 034 023 023 022 021 035  036 035 032
051 038 024 037 042 020 018 022 024 028 026  016 029 028
036 052 033 053 051 022 032 030 024 019 017  027 028 039
038 049 034 053 079 012 010 028 019 012 022  043 057 051
037 042 016 068 075 016 035 025 029 024 033  060 057 079
044 057 018 081 111 029 027 037 032 057 024  042 074 067
060 103 079 086 078 022 053 043 083 061 044  070 076 087
121 129 126 114 113 038 066 060 093 063 121  153 178 226
078 106 125 119 176 153 137 166 123 117 140  144 236 224
069 114 134 127 201 159 077 143 184 163 165  213 188 172
049 112 129 074 120 089 024 051 079 054 061  045 035 041
064 117 138 056 064 087 033 044 039 030 024  023 013 017
082 138 140 118 039 034 049 059 059 046 042  048 035 071
```

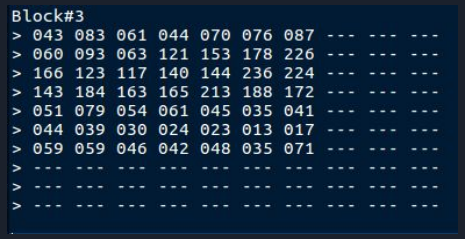
8 X 8 Block Alignment In shared data, Kernel of size 3 (1 neighbor in each direction)











Algorithm 1

Given a picture of size n by m, and a convolution kernel of size k by k. i= floor(sqrt(k)) neighbors

- For block sizes bY, bX, initialize the GPU kernel with size bY+2i, bX+2i. Allowing to copy the ith neighbor if valid.
- Initialize some dynamic shared memory for each block of size blockDim.x by blockDim.y
- Use a constant mapping function that maps index in the shared memory to an index in the input array
- In parallel copy from global to shared memory
- For thread within a block, compute the convolution of each entry and its i neighbors, with the kernel matrix
- Compute the convolution O(k^2)
- The algorithm therefore takes O(k^2) but...

Algorithm 1

- We can do better with the Convolution step, ...theoretically
  - Use Dynamic Parallelism
  - Each thread within a block spawns k^2 threads in a single block
  - Pass pixel input to their handling kernel along with the kernel matrix
  - Each sub-thread to do a single multiplication and sync with the rest
  - In O(log(k^2)), do a Reduce algorithm to compute the convolution and thread0 to write the result to the first element of the array.
  - O(log(k^2)), Much better than sequential convolution
- Drawback
  - CUDA does not allow block shared memory to be passed to child kernels
  - Solution Allocate Global memory, or malloc and memcpy the array. Not a good idea.
  - GPU are high throughput devices, Multiple allocation and memory copies become exponentially expensive, even on small size problems

Algorithm 2 Dynamic Parallelism

- Uses global memory and shared memory
- For the same problem size, and kernel size do:
  - Create enough blocks with overall size the size of the input array
  - Each threat to work on a single entry in the input array
  - Each thread to spawn a kernel of block size that of the kernel matrix
  - In log(k^2) compute the convolution and store the result directly to the global output array
    - Synchronization needed only in the sub-blocks in computing the parallel reduce algorithm
- Performance remains slower than algorithm 1 by a factor of 100
  - Each thread, or subthread read from input array and writes to the output array. However the true cost is the global memory reads that each sub-thread has to do in order to fill their shared sub-block memory
  - For 14*14 image, 3*3 kernel matrix, 196 1st level threads are spawned, 1764 second level threads, 1764 reads from possibly different memory banks.

Algorithm 2 Dynamic Parallelism

- For 1920*1080 image, 3*3 kernel matrix, 2.1M 1st level threads are spawned, 18M second level threads, 18M reads from possibly different memory banks.
- Becomes out of control

Synchronization

- Algorithm 1 requires the least amount of synchronization, which is highly desirable for speeding up.
- Algorithm 1 needs only synchronization for shared memory ready. After which, can move on and write to the output array.
- Algorithm 2 requires quire more synchronization, although the synchronization happens more at the sub kernel level for data ready and parallel reduce, which adds the cost of this algorithm

Important about Convolution Kernels

- Requires square kernel matrices of odd size
- Multiple kernels matrices can be convoluted to form a single kernel before applying it to the input image.
- Kernels have to be normalized to avoid modifying the brightness of the original image

Conclusion and Future Work

- It has been proven in this project that designing an algorithm for parallel computing requires more consideration than the theory of time and work complexities.
- The memory alignment and hierarchy plays a role just as important.

- Research further more on Dynamic Parallelism and its applicability
- Continue on this project to provide a Graphical User Interface
- Implement Steganography and Watermarking with a GPU
  - Steganography the practice of concealing a file, message, image, or video within another file, message, image, or video

- Digital Image Processing is an interesting field, fun, …. until one pixel you cannot get rid of appears.

DEMO

THANKS