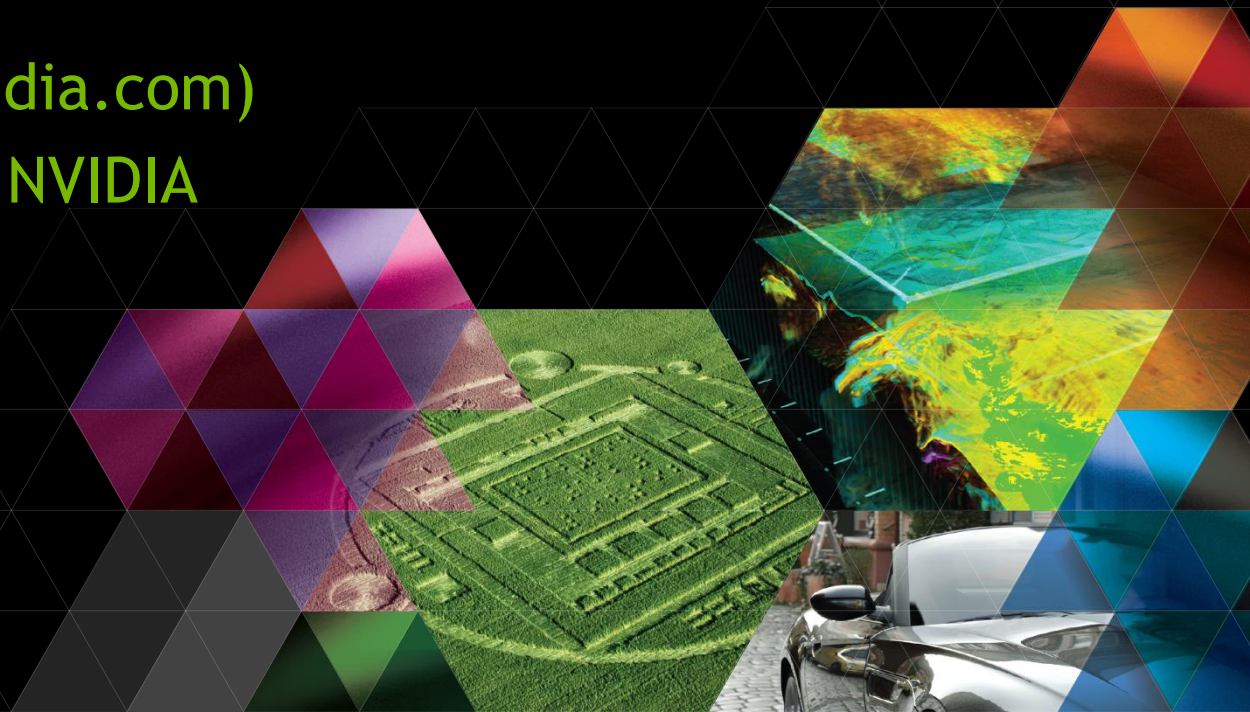
GPU TECHNOLOGY CONFERENCE

NVIDIA

HIGH PERFORMANCE EDGE-PRESERVING FILTER ON GPU

Jonas Li (jonasl@nvidia.com)
Compute Architect, NVIDIA



GPU TECHNOLOGY CONFERENCE

AGENDA

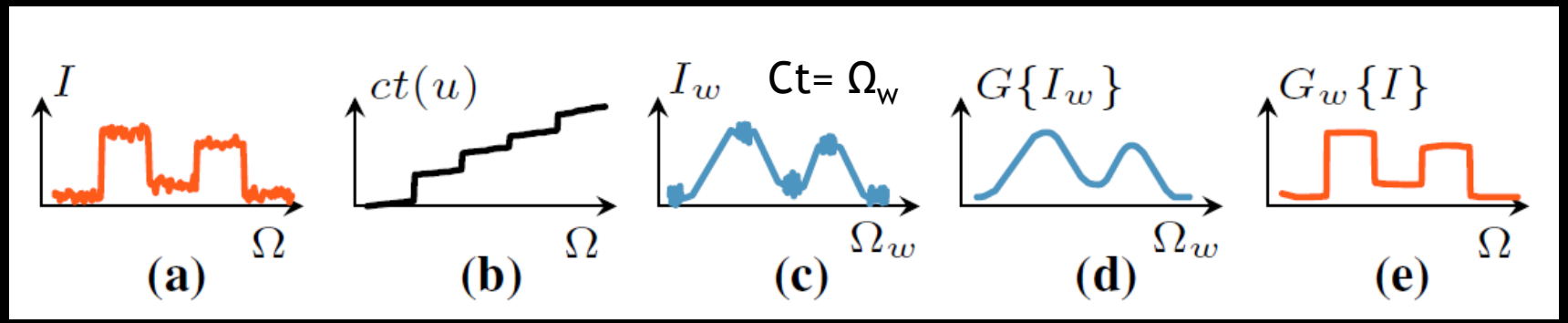
- Algorithm basis
- Optimization on GPU
- Performance data
- Demo

ALGORITHM BASIS

- Based on state-of-art "domain transform[1]"
  - From **R^5(X,Y,R,G,B)** to **R^2(XY,RGB)**, distances preserved
  - (X,RGB)->ct(x)
  - (Y,RGB)->ct(y)
  - 1D smoothing on ct(x) and ct(y) ≡ edge preserving on 2D color image

ct(x) is accumulation of each L_1 distance of (x, I(x))

$$ct(u) = \int_0^u 1 + |I'(x)|dx$$



[1]: http://www.inf.ufrgs.br/~eslgastal/DomainTransform/

ALGORITHM BASIS

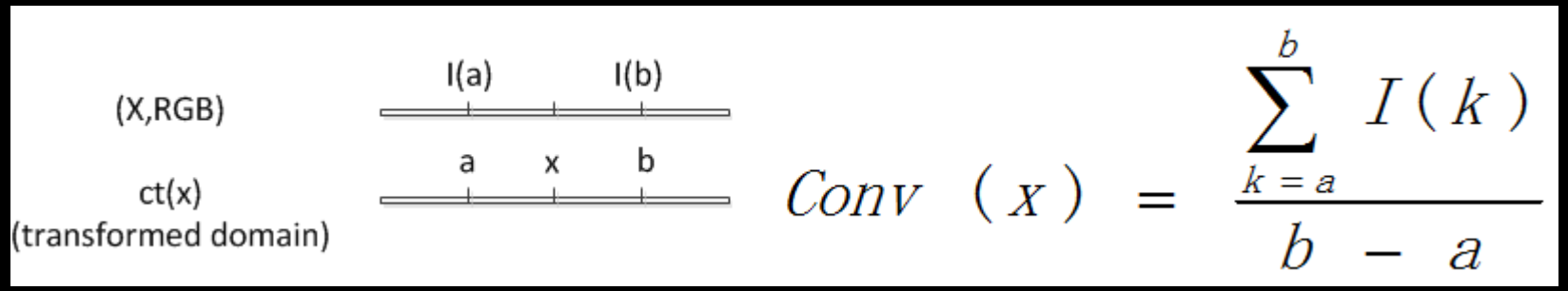
- Based on state-of-art "domain transform[1]"
  - From **R^5(X,Y,R,G,B)** to **R^2(XY,RGB)**, distances preserved
  - (X,RGB)->ct(x)
  - (Y,RGB)->ct(y)
  - 1D smoothing on ct(x) and ct(y) ≡ edge preserving on 2D color image

ct(x) is accumulation of each L_1 distance of (x, I(x))

[1]: http://www.inf.ufrgs.br/~eslgastal/DomainTransform/

ALGORITHM BASIS

- GPU implementation
  - Normalized convolution is GPU friendly
  - Implemented on Tegra K1
  - 2 main phases
    - Integral image(calculate ct(x), ct(y))
    - Normalized convolution(binary search, matrix transposition)

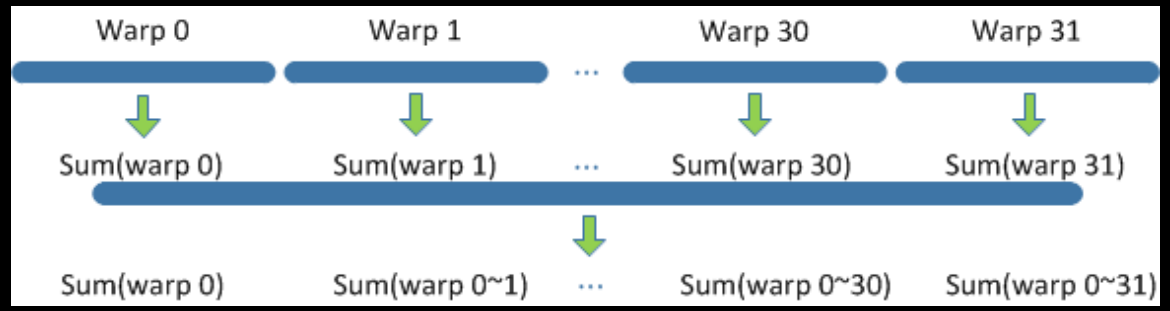


$$Conv(x) = \frac{\sum_{k=a}^{b} I(k)}{b-a}$$

1ST PHASE: INTEGRAL IMAGE

- Naïve implementation
  - Large thread block
  - Warp-wide shuffle
  - Shared memory
- Issues
  - Warp synchronization

```
partial_sum = integral(thread_value);
SMEM <- partial_sum;
sync();
if(warp_id == 0)
    warp_sum = integral(partial_sum);
SMEM <- warp_sum;
sync();
If(warp_id != 0)
     thread_value += warp_sum;
sync();
output();
```

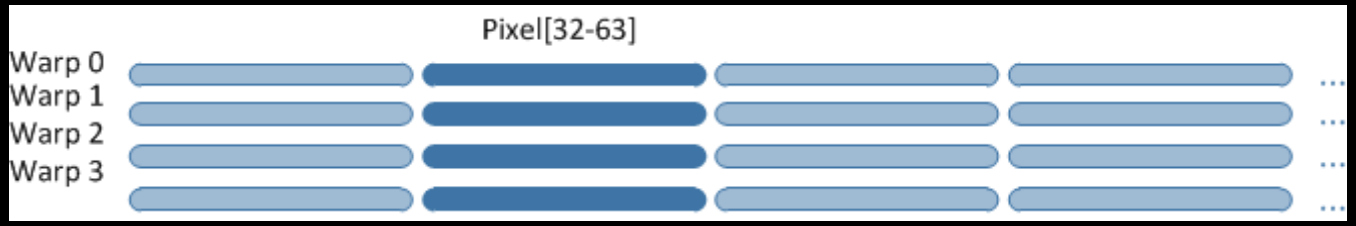


1ST PHASE: INTEGRAL IMAGE

- Optimized implementation
  - Eliminate warp synchronization
  - Data prefetching
  - **2.7x** speedup vs. naïve code
  - **95%** of peak performance

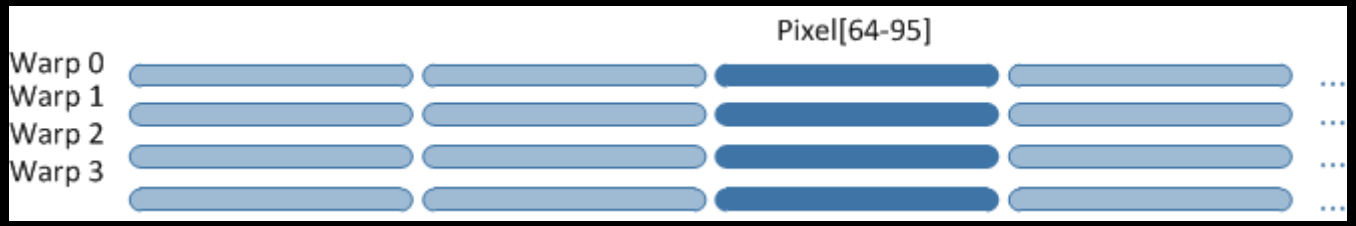
1ST PHASE: INTEGRAL IMAGE

- Optimized implementation
  - Eliminate warp synchronization
  - Data prefetching
  - **2.7x** speedup vs. naïve code
  - **95%** of peak performance



1ST PHASE: INTEGRAL IMAGE

- Optimized implementation
  - Eliminate warp synchronization
  - Data prefetching
  - **2.7x** speedup vs. naïve code
  - **95%** of peak performance



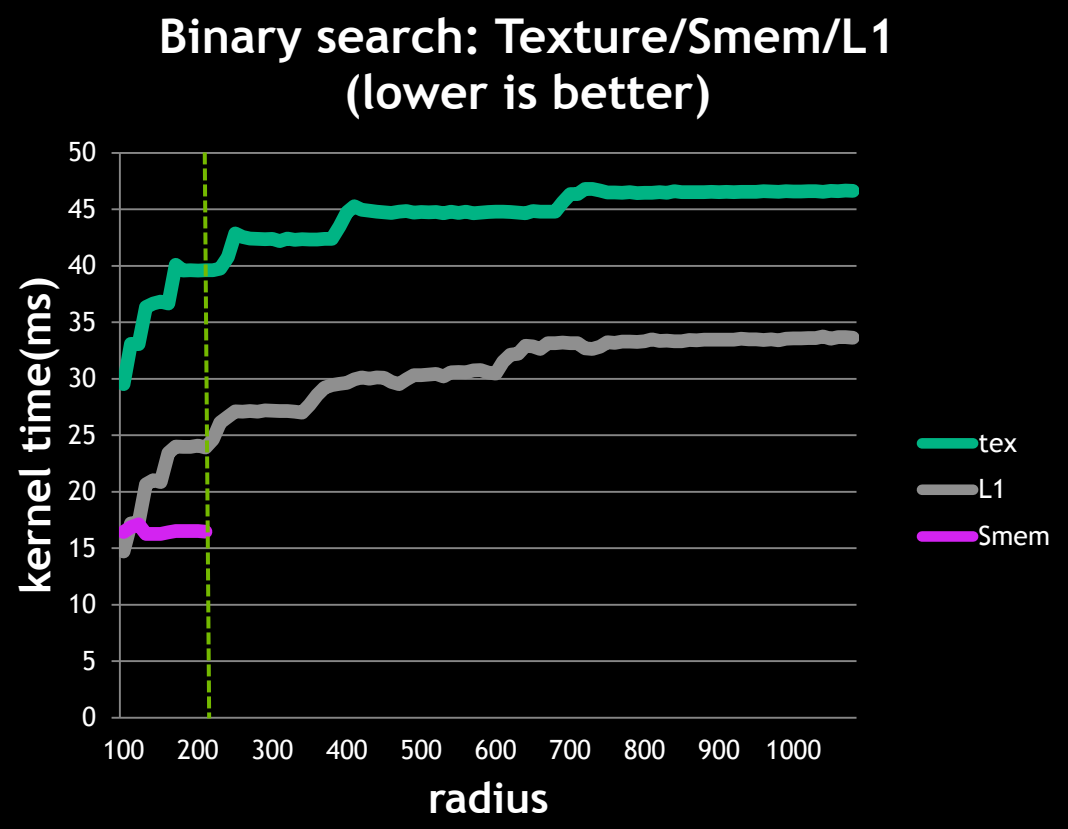
2ND PHASE: NORMALIZED CONVOLUTION

- Per-thread binary search
  - Partially overlapped range
  - Still highly divergent
  - Data dependency
- Matrix transposition
  - Bank conflicts

```
while (right > left) {
    idx = (right + left)/ 2;
    v = input(idx);
    if (v > val) {
        right = idx;
    } else {
        left = idx + 1;
    }
}
```

BINARY SEARCH

- Texture
  - Seems natural to handle divergence
  - Longer latency
  - 2D cache locality
- Shared memory
  - Data copy/refresh overhead
  - Limited size per thread
- L1 Cache
  - As fast as shared memory
  - Hardware managed data refresh

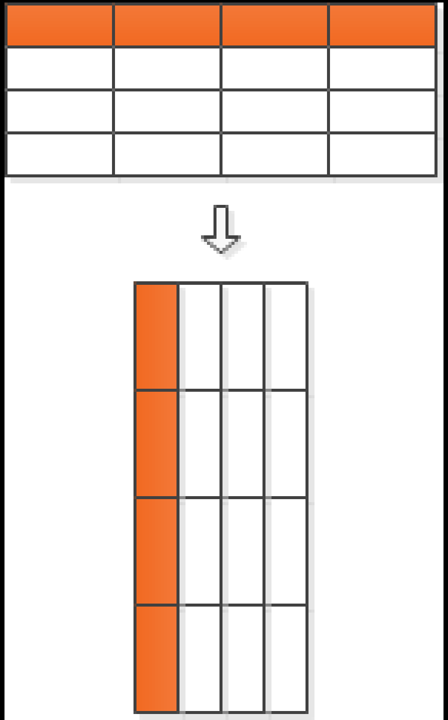


MATRIX TRANSPOSITION

- Previous implementation[1]
  - Transpose in shared memory
  - Tile padding to avoid bank conflict
- Our implementation
  - Transpose in shared memory
  - No bank conflict
  - No tile padding

[1]: http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/#shared-memory-in-matrix-multiplication-c-aa

MATRIX TRANSPOSITION

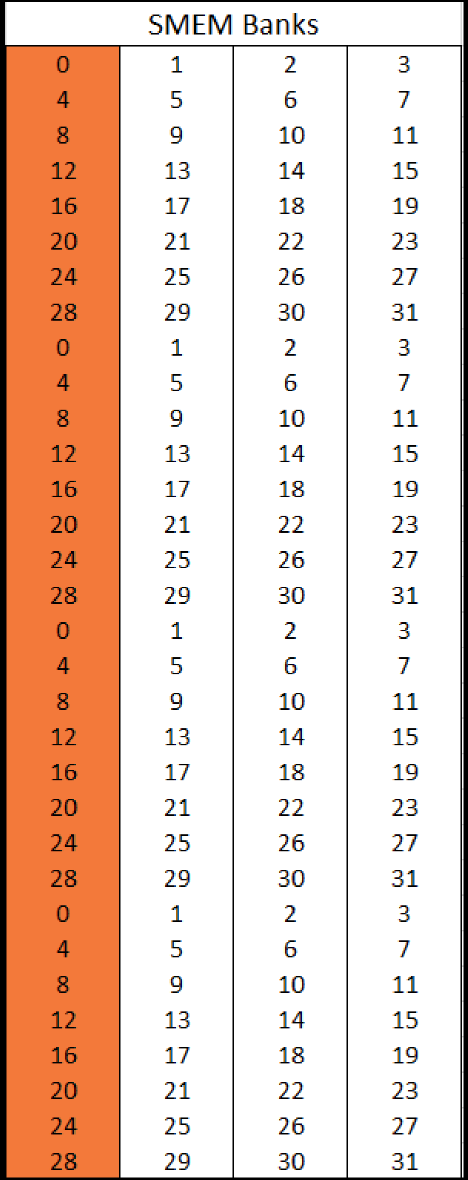


4 warps per block

4 bytes per pixel

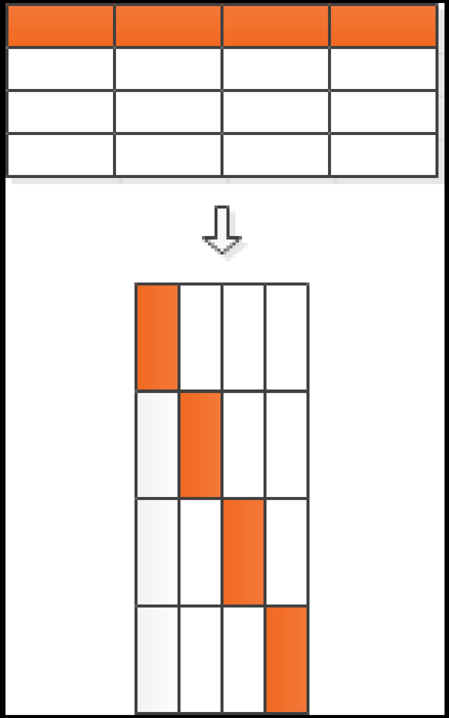
Naïve implementation:

4-way bank conflict in shared memory



SMEM Banks			
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31

MATRIX TRANSPOSITION



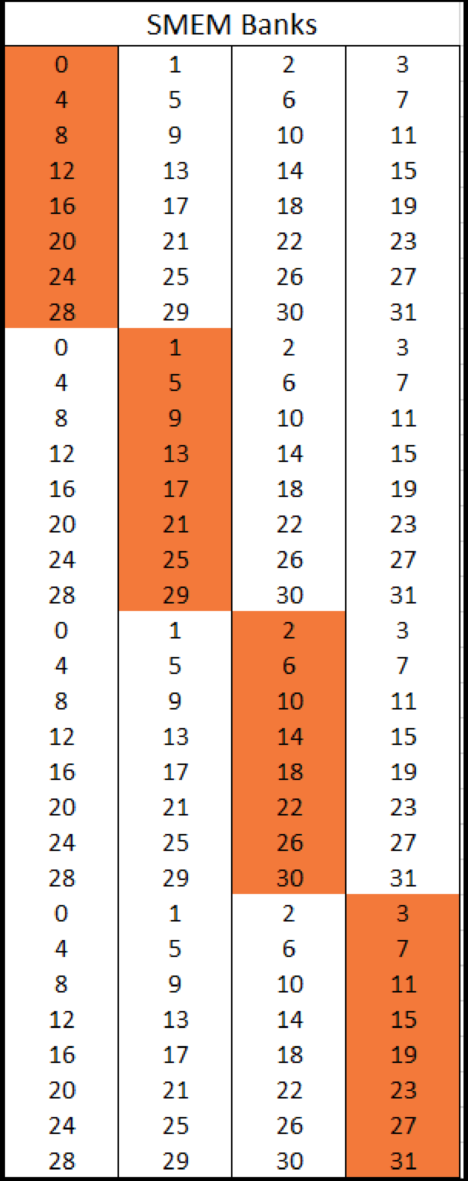
4 warps per block

4 bytes per pixel

Our implementation:

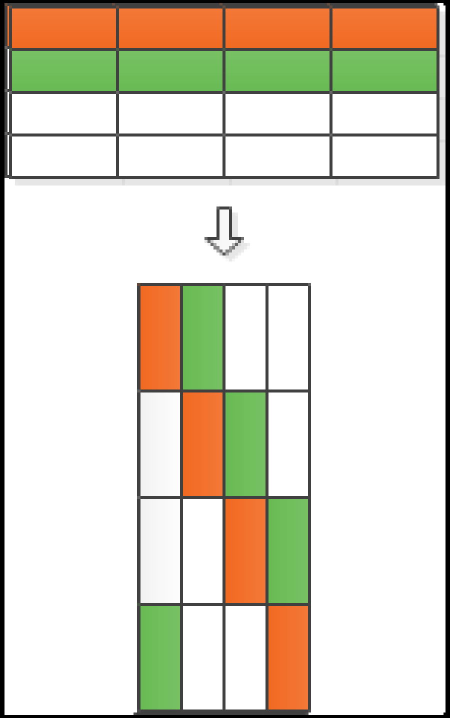
Store: No bank conflict

Load: No bank conflict



SMEM Banks			
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31

MATRIX TRANSPOSITION



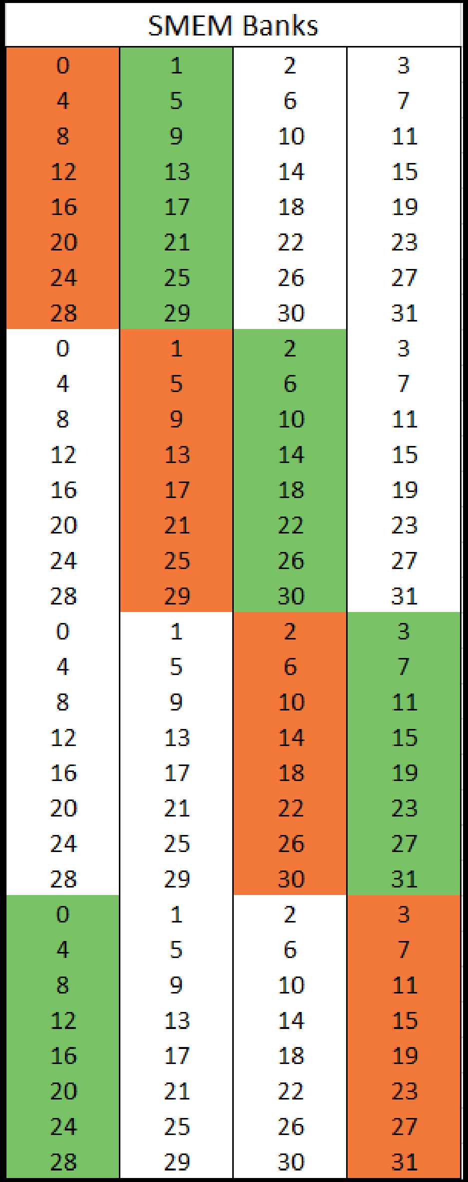
4 warps per block

4 bytes per pixel

Our implementation:

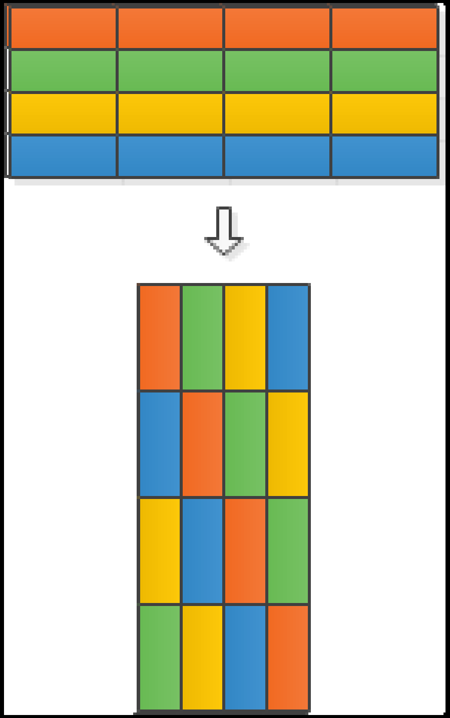
Store: No bank conflict

Load: No bank conflict



SMEM Banks			
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31

MATRIX TRANSPOSITION



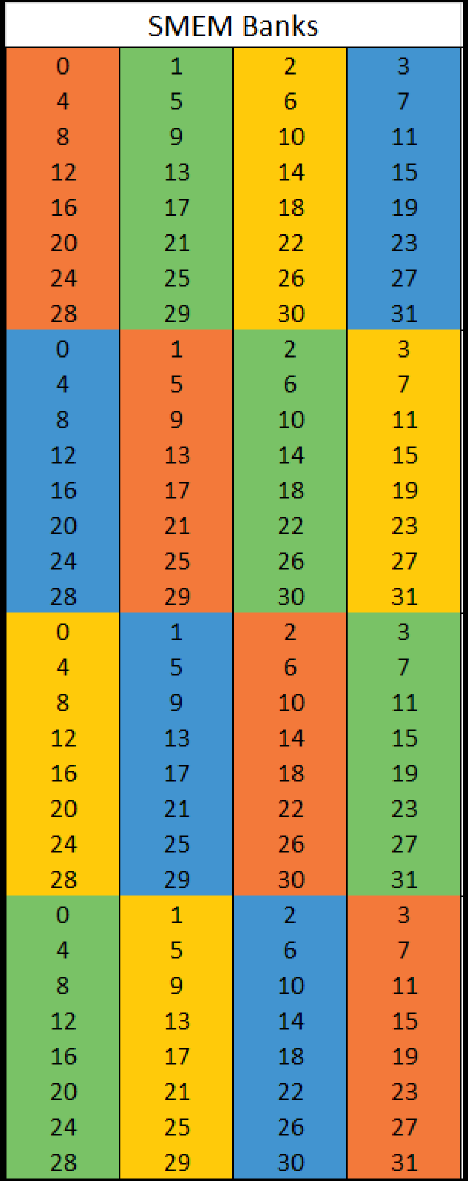
4 warps per block

4 bytes per pixel

Our implementation:

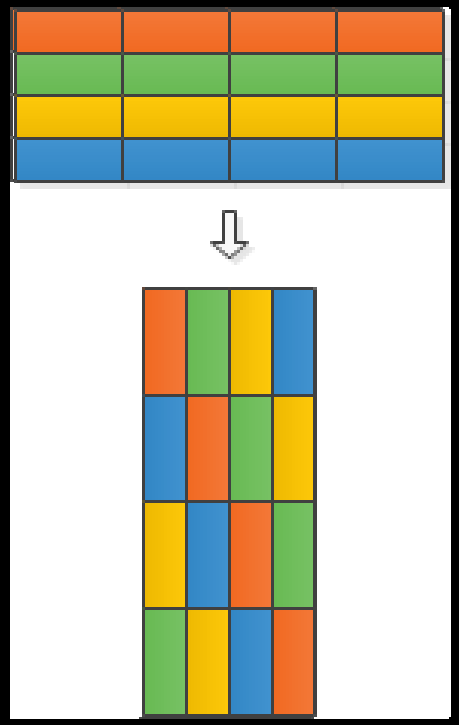
Store: No bank conflict

Load: No bank conflict



SMEM Banks			
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31

MATRIX TRANSPOSITION



4 warps per block

4 bytes per pixel

Our implementation:

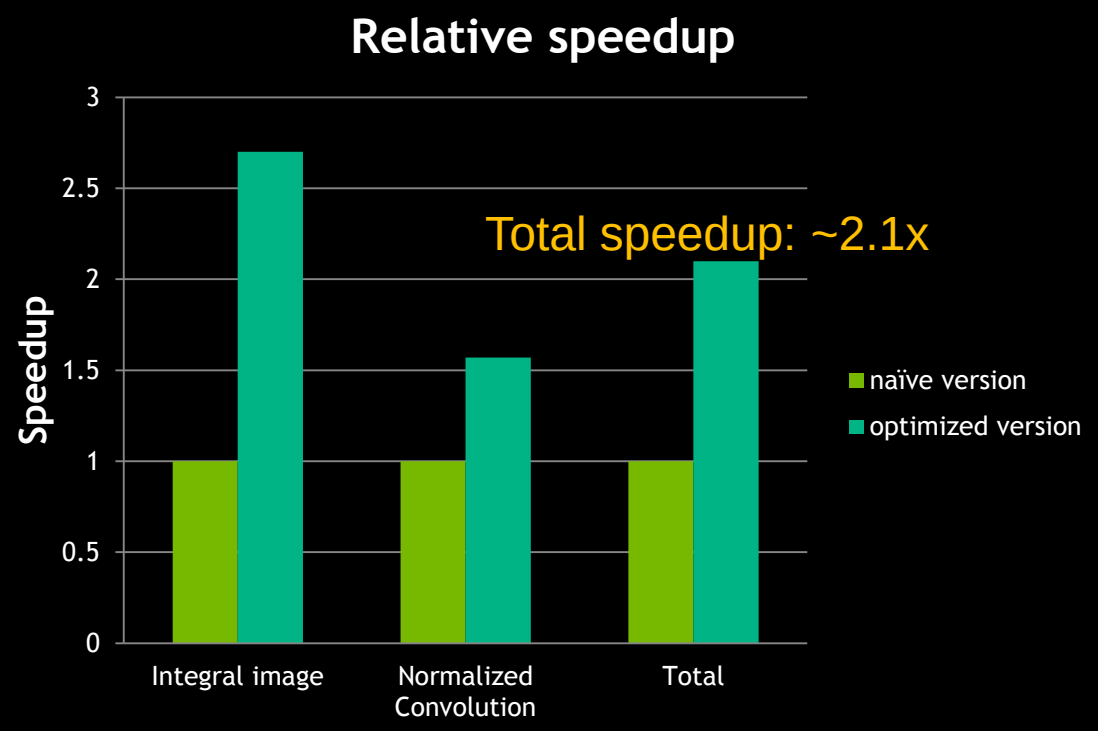
Store: No bank conflict

Load: No bank conflict

SMEM Banks				
0	1	2	3	warp 0
4	5	6	7	
8	9	10	11	
12	13	14	15	
16	17	18	19	
20	21	22	23	
24	25	26	27	
28	29	30	31	
0	1	2	3	warp 1
4	5	6	7	
8	9	10	11	
12	13	14	15	
16	17	18	19	
20	21	22	23	
24	25	26	27	
28	29	30	31	
0	1	2	3	warp 2
4	5	6	7	
8	9	10	11	
12	13	14	15	
16	17	18	19	
20	21	22	23	
24	25	26	27	
28	29	30	31	
0	1	2	3	warp 3
4	5	6	7	
8	9	10	11	
12	13	14	15	
16	17	18	19	
20	21	22	23	
24	25	26	27	
28	29	30	31	

PERFORMANCE DATA

- Performance comparison with naïve version
- Tegra K1, 1920*1080,BGR color video
  - Naïve version: 15.52fps
  - Optimized version: 32.59fps



SUMMARY

- A real-time edge-preserving filter on GPU is presented
- Fast integral without warp synchronization
- L1-based in-place binary search
- Efficient matrix transpose scheme

DEMO