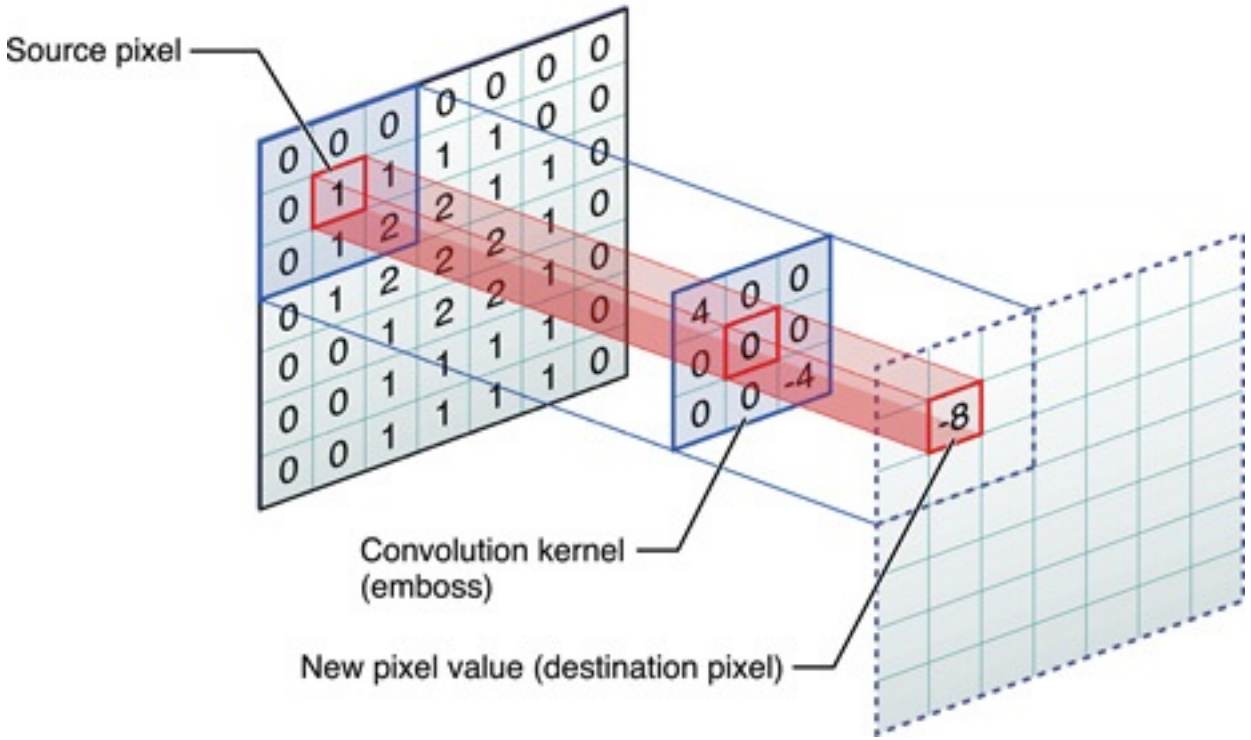
CONVOLUTION CUDA

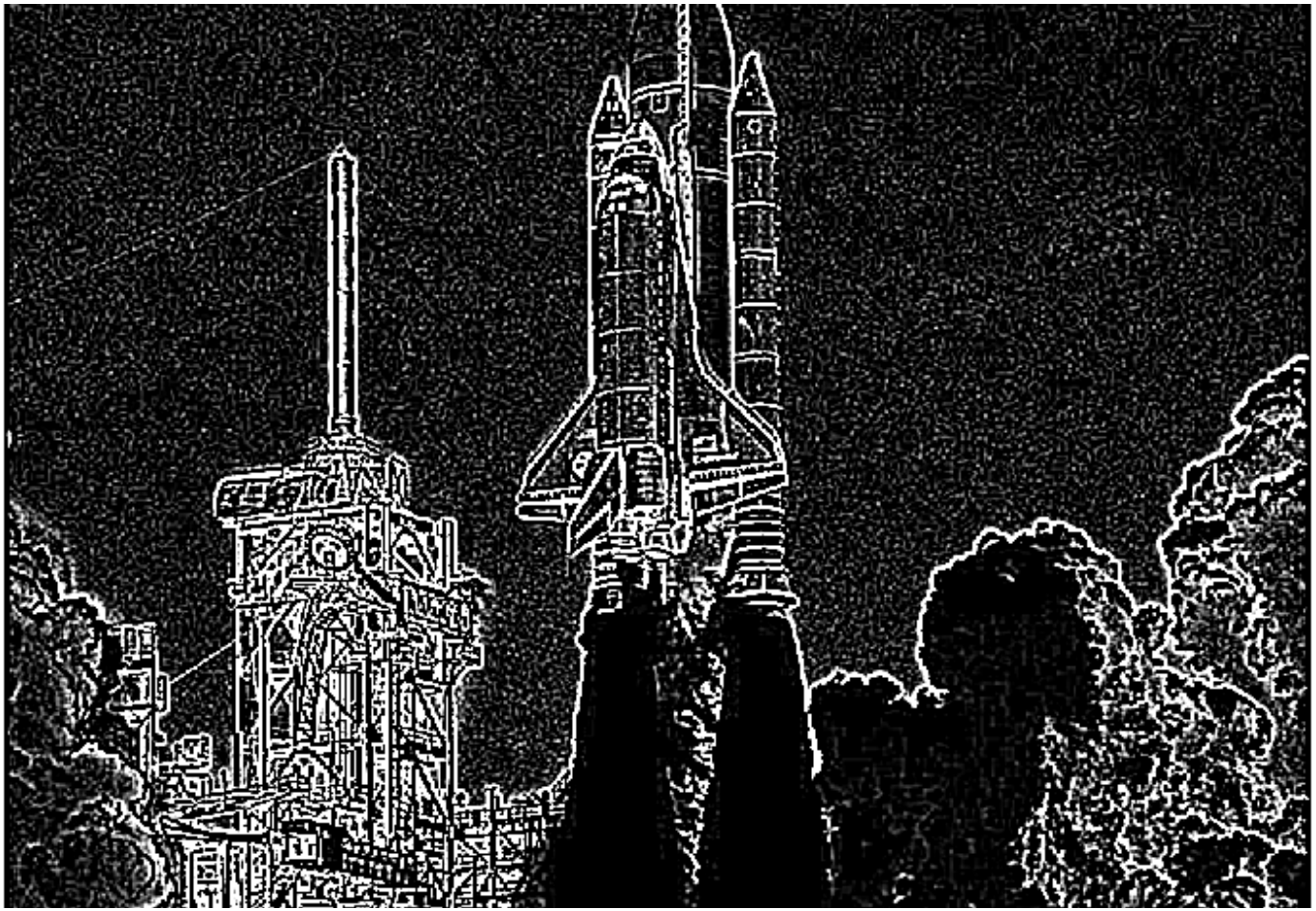
VINCENT PASQUIER

GROUP MEMBER: DORIAN GAMBIN

CONVOLUTION ?



RESULT



GRAYSCALE ?

Color transformation to gray values [0, 255]

Lightness

- (max(R, G, B) + min(R, G, B)) / 2

Average

- (R + G + B) / 3

Luminance

- 0.21 R + 0.71 G + 0.07 B

GRAYSCALE EXAMPLE

NAÏVE IMPLEMENTATION

Matrix in global memory

Kernel in global memory

Access each pixel one-by-one

ALGORITHM OPTIMIZATIONS

1. **Use texture to handle border**
2. **Access values in corners then cross and sum**
3. **Store kernel in constant memory**
4. **Use cuda instruction to multiply**
    1. __[u]mul24(a, b)

TRANSFERT OPTIMIZATION

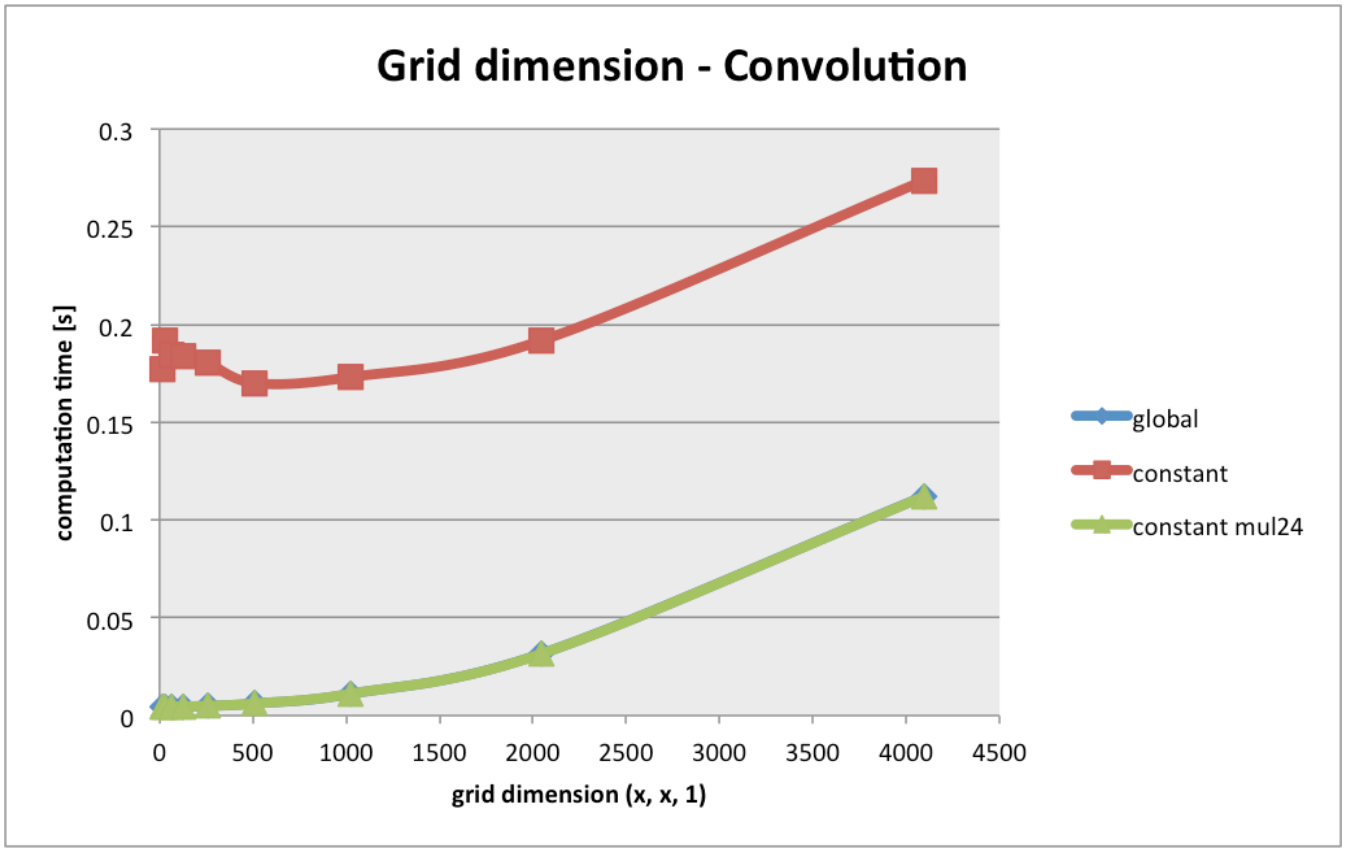
Use DMA to transfert images

Analogy

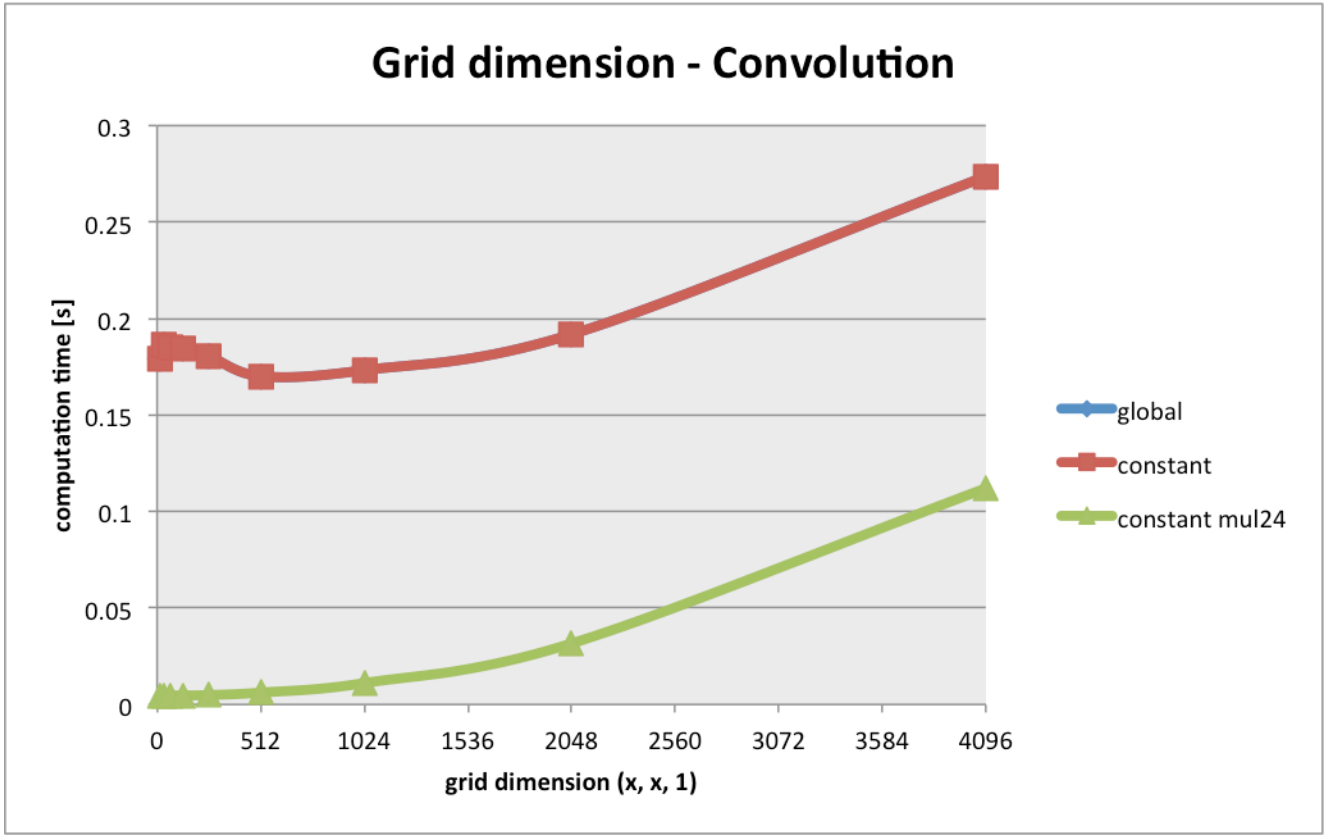
- **Theatre wardrobe**
- **Swimming pool**

```
size_t data_size = getH () * getW () * sizeof(uchar3);
HANDLE_ERROR( cudaHostAlloc ((void**) &_data, data_size,
cudaHostAllocDefault ) );
matCaptureSrc =  Mat ( getH (), getW (), CV_8UC3, _data );
```

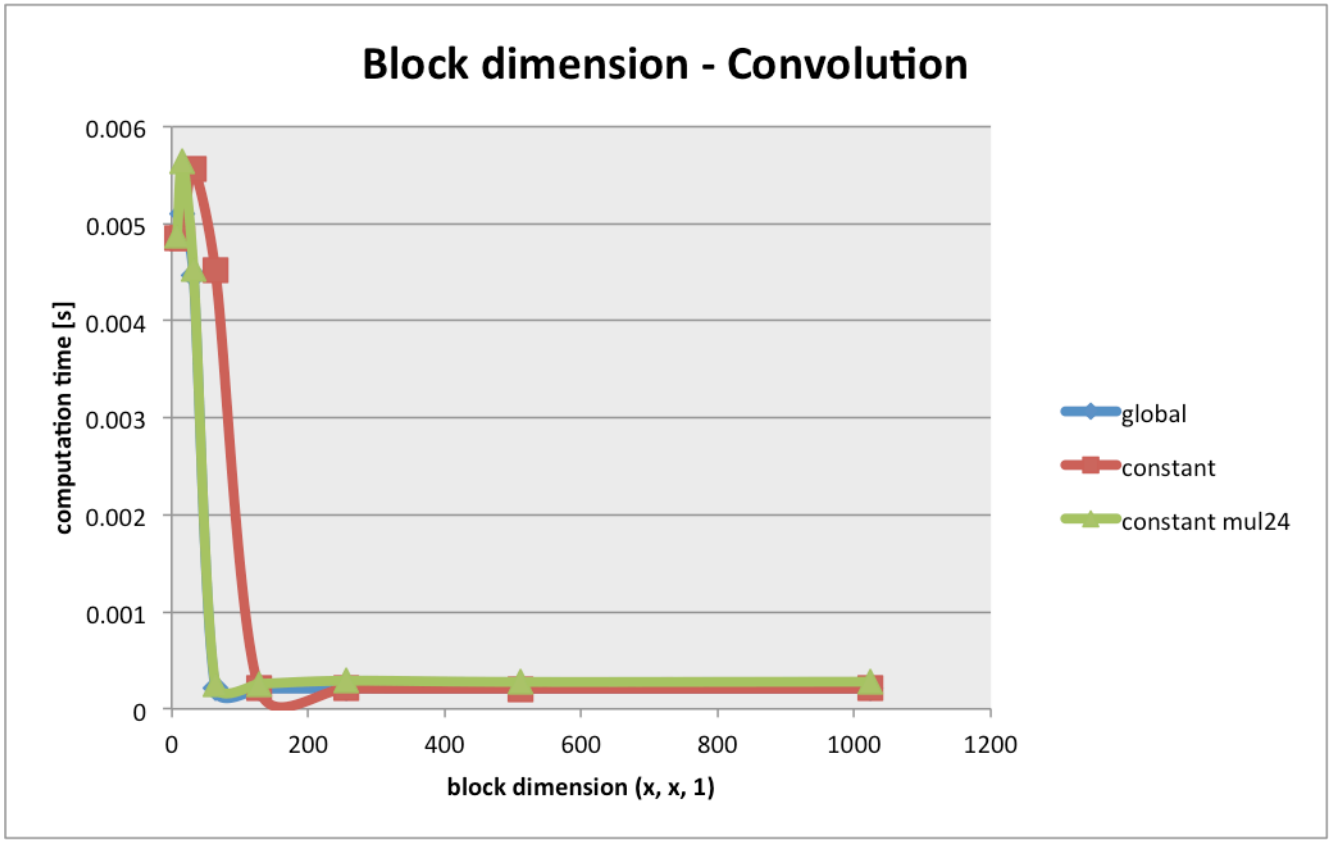
RESULTS


Grid dimension - Convolution
computation time [s]
0.3
0.25
0.2
0.15
0.1
0.05
0
0
500
1000
1500
2000
2500
3000
3500
4000
4500
grid dimension (x, x, 1)
global
constant
constant mul24

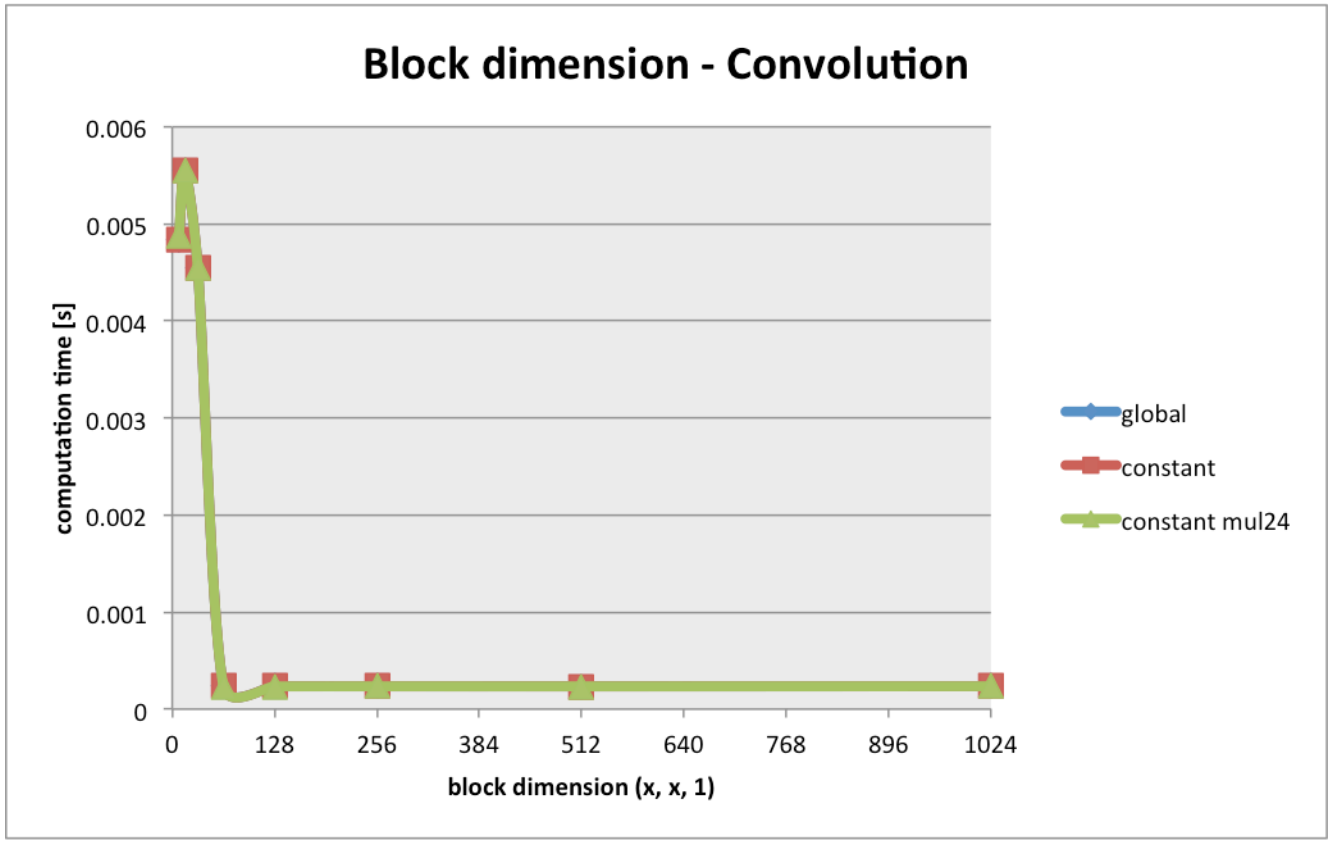
RESULTS


Grid dimension - Convolution
computation time [s]
0.3
0.25
0.2
0.15
0.1
0.05
0
0
512
1024
1536
2048
2560
3072
3584
4096
grid dimension (x, x, 1)
global
constant
constant mul24

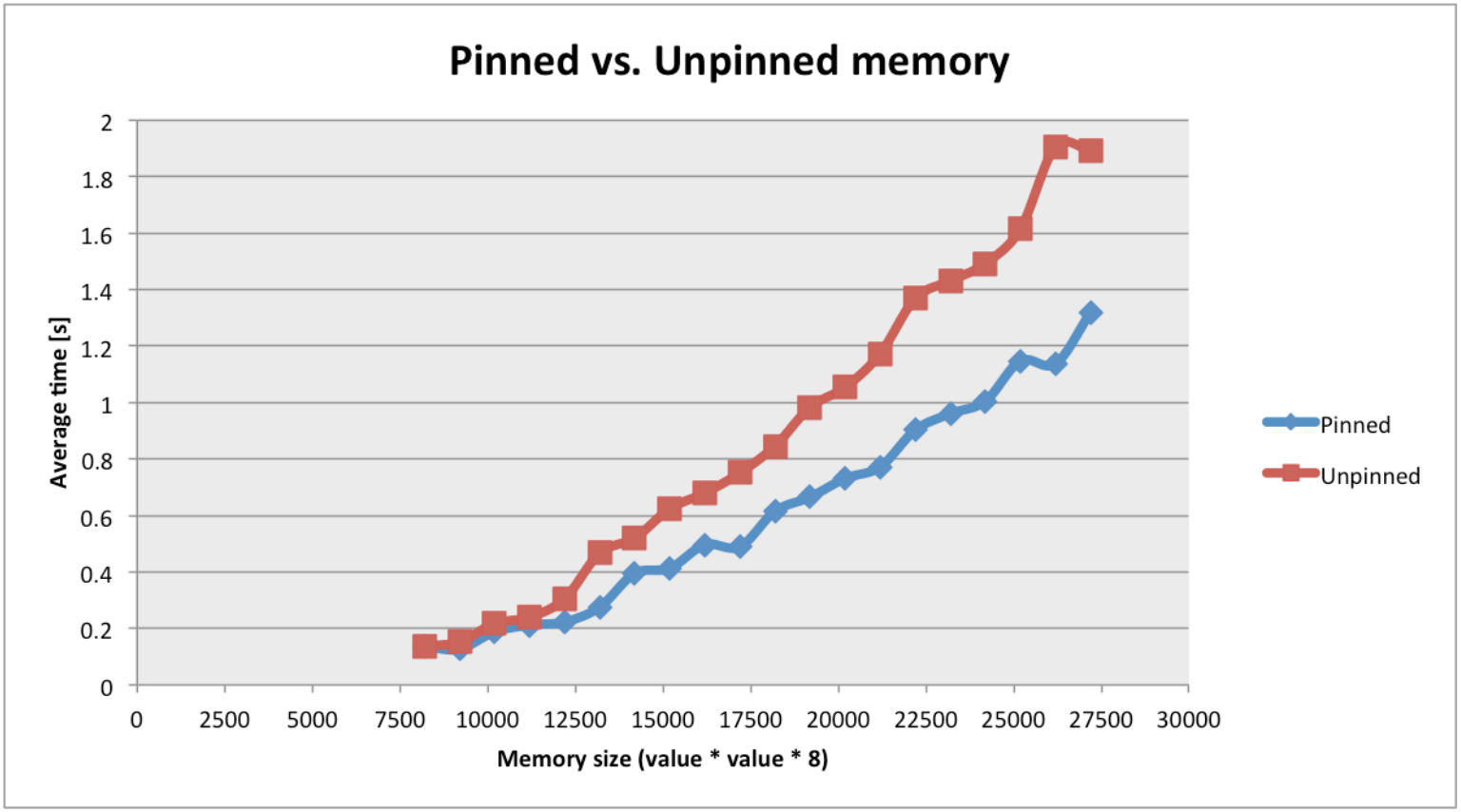
RESULTS


Block dimension - Convolution
computation time [s]
0.006
0.005
0.004
0.003
0.002
0.001
0
0
200
400
600
800
1000
1200
block dimension (x, x, 1)
global
constant
constant mul24

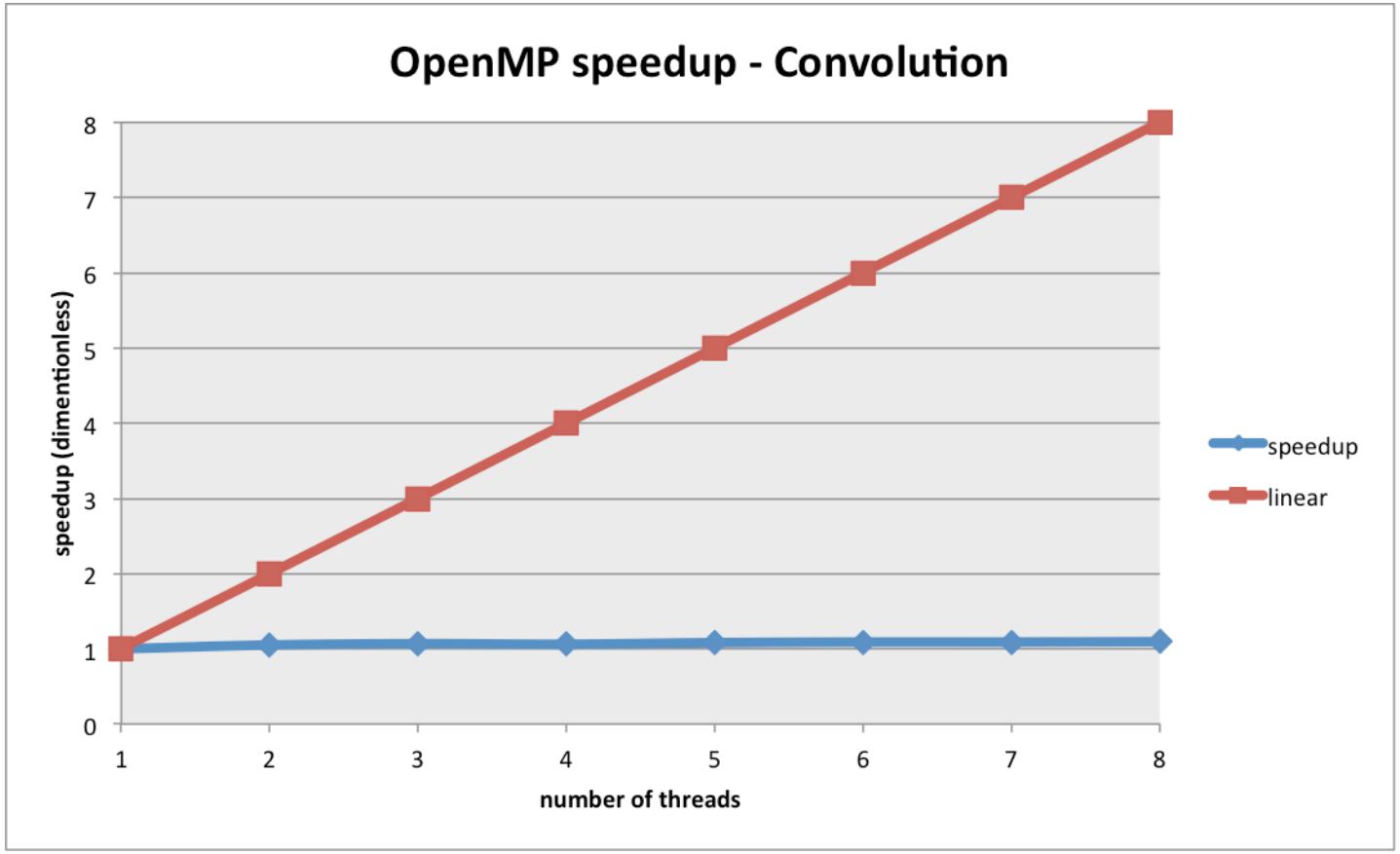
RESULTS


Block dimension - Convolution
0.006
0.005
0.004
0.003
0.002
0.001
0
computation time [s]
0
128
256
384
512
640
768
896
1024
block dimension (x, x, 1)
global
constant
constant mul24

RESULTS DMA


Pinned vs. Unpinned memory
Average time [s]
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
2500
5000
7500
10000
12500
15000
17500
20000
22500
25000
27500
30000
Memory size (value * value * 8)
Pinned
Unpinned

OPENMP RESULTS



FUTURE OPTIMIZATIONS

Kernel separation

- **Algorithm complexity reduced**
- **9*9 instructions**
- **9+9 instructions**

```
[u,s,v] = svd(H);
s = diag(s);
if sum(s > eps('single'))==1
   h1 = u(:,1)*s(1);
   h2 = v(:,1)';
else
   error('Kernel not separable!')
end
q = h1*h2-H;
q = abs(q(:))/max(abs(H(:)));
max(q) < eps('single')
```

Sadly, not separable ☹

KERNEL SEPARATION

```
k = -1     0     1
    -2     0     2
    -1     0     1

v =  1
     2
     1

u =  -1     0     1
```

IMPLEMENTATION TEXTURE / CONSTANT

```
float sum = 0.0f;
 for ( uint_32 v = 1; v <= kHalf; v++ ) {
   for ( uint_32 u = 1; u <= kHalf; u++ ) {
     sum += k_KERNEL[center + ( v * k ) + u] * tex2D ( texBWImage, j + v, i + u ).x;
     sum += k_KERNEL[center + ( v * k ) - u] * tex2D ( texBWImage, j + v, i - u ).x;
     sum += k_KERNEL[center - ( v * k ) + u] * tex2D ( texBWImage, j - v, i + u ).x;
     sum += k_KERNEL[center - ( v * k ) - u] * tex2D ( texBWImage, j - v, i - u ).x;
   }
 }
 for ( int32_t u = -k / 2; u < k / 2; u++ ) {
   sum += k_KERNEL[center + u] * tex2D ( texBWImage, j, i + u ).x;
   sum += k_KERNEL[center + k * u] * tex2D ( texBWImage, j + u, i ).x;
 }
 // Center computed twice.return sum;
 sum += ( k_KERNEL[center] * tex2D ( texBWImage, j, i ).x );
```

FUTURE OPTIMIZATIONS

Divide & conquer

Send image to all graphic cards

Sadly, the battle is lost ☹

CONCLUSION

- VideoCapture: RGB instead of RBGA
- DMA is faster
- CUDA is perfect for this problem
- Optimization performances

PERSPECTIVES