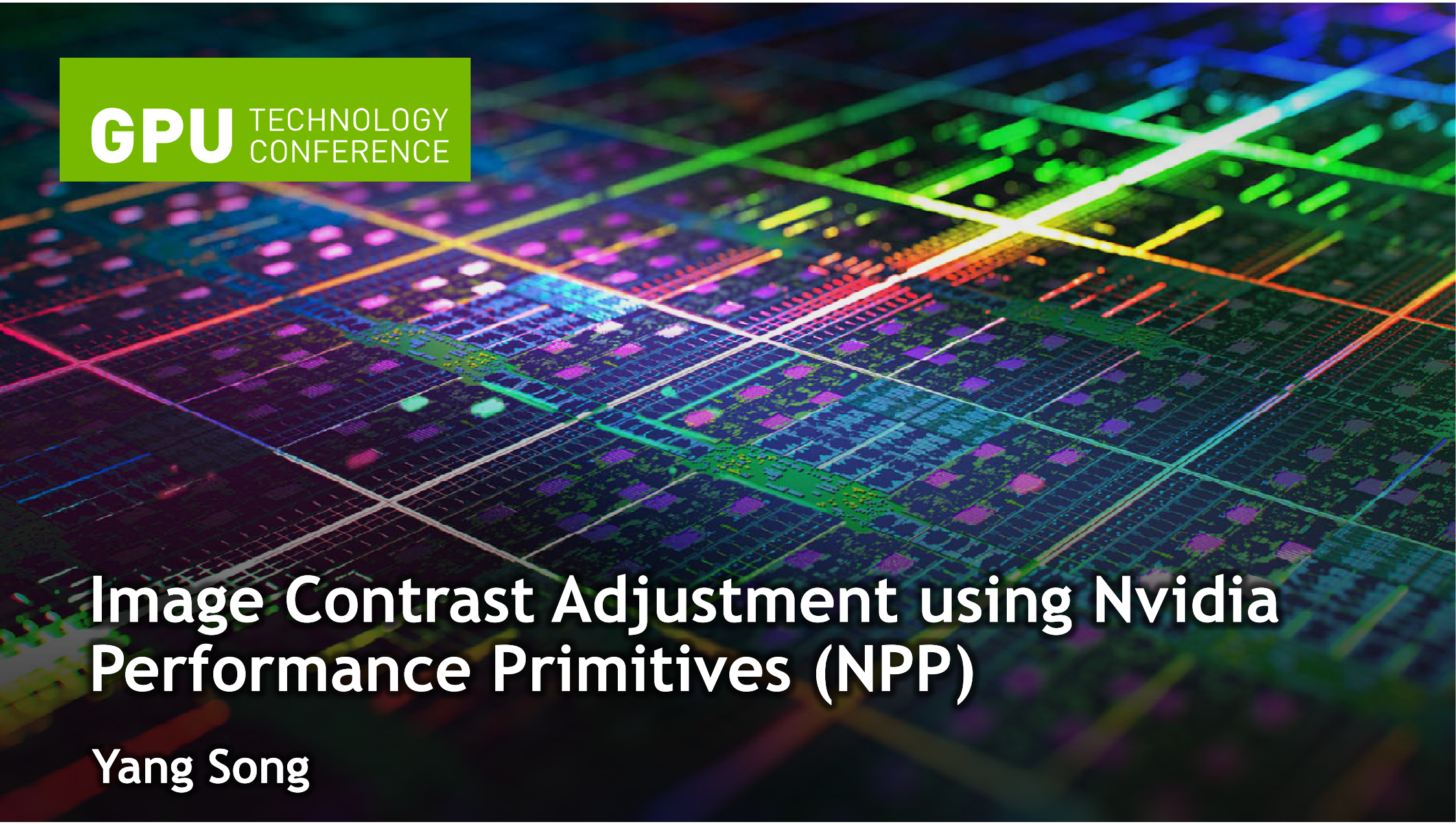
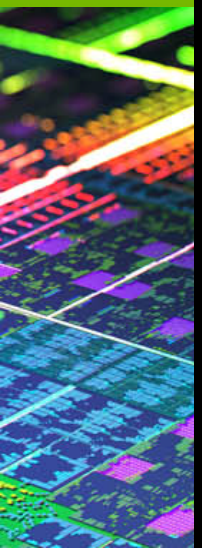


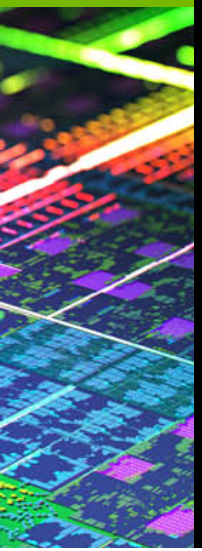
Image Contrast Adjustment using Nvidia Performance Primitives (NPP)

Yang Song



Outline

- NPP Introduction
- Problem Statement
- Solution and Result
- Further Reading/Resources



What is NPP?

- A library of GPU-accelerated image, signal and video processing functions.
- Key features:
  - Arithmetic and Logical Operations
  - Statistical Operations
  - Filter Functions, etc.
- Performance
  - 5x ~ 10x than CPU-only implementation.
- Amount
  - Around 4000 in CUDA 5.0.

Problem Statement

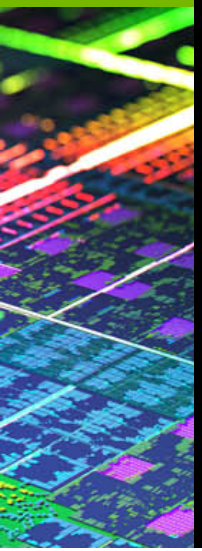
- Hazy source image lacks contrast.
- Adjustment algorithm (8-bit image):
  - Offset and scale image, such that darkest pixel is mapped to 0 and brightest pixel is mapped to 255

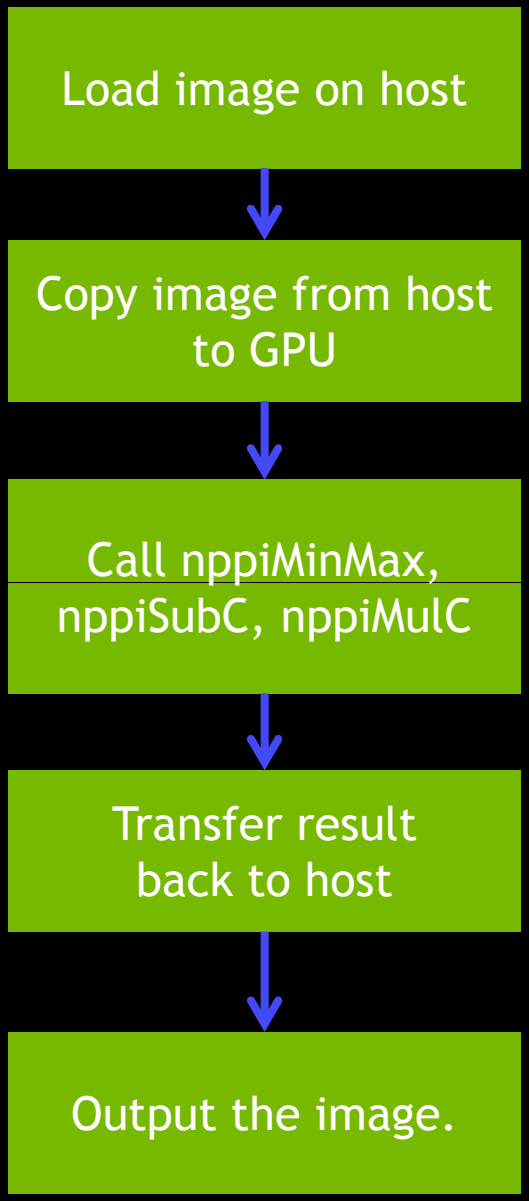
$$pDst(i,j) = 255 \times \frac{pSrc(i,j) - nMin}{nMax - nMin}$$

- Three operations needed:
  - MinMax, Subtract and Multiply

Good News!
NPP has them all ☺

Solution

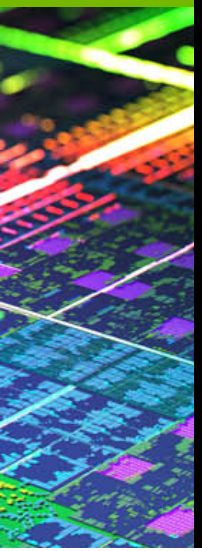




You have to fill in this part.

NPP Image Representation

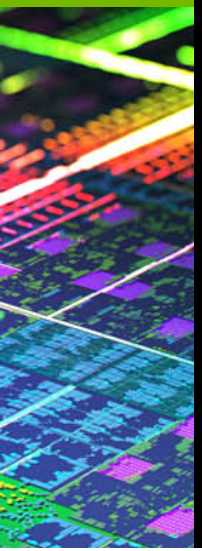
- Image is represented by two parameters:
  - pSrc: pointer to the first pixel of the image
  - nSrcStep: number of bytes between successive rows

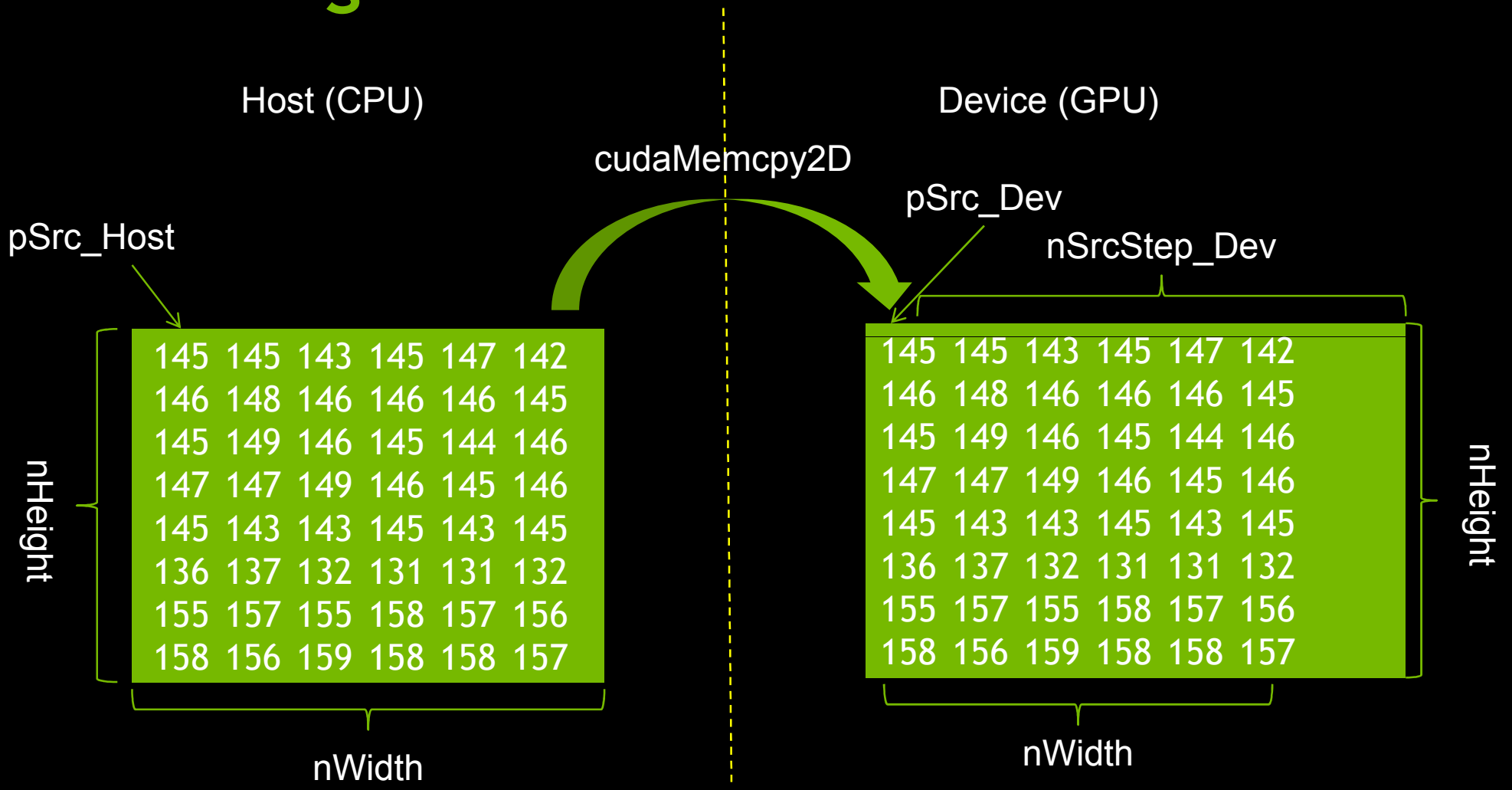


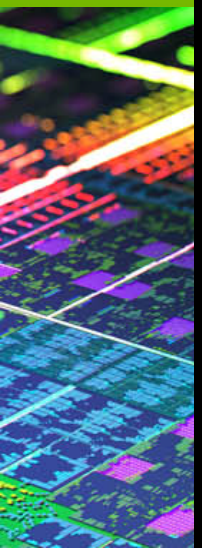
Function Naming

- nppiMulC_8u_C1IRSfs
  - npp
  - i: image module (s: signal module)
  - MulC: primitive name (Add, Sum, etc.)
  - 8u: data type of the image (16u, 32s, 32f, 64f, etc.)
  - C1: single channel (C3R, C4R, AC4R, etc.)
  - I: in-place (out-of-place if not specified)
  - R: work on ROI (region of interest)
  - Sfs: allow result scaling

Read Image

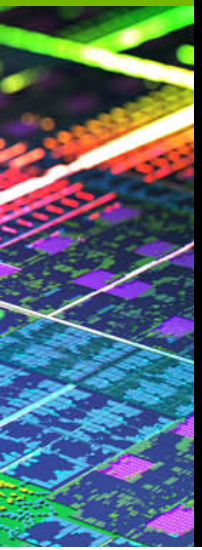






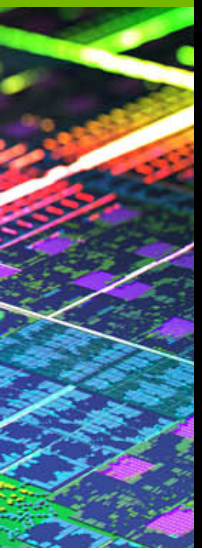
NPP scratch buffer

- nppiMinMax_8u_C1R needs device buffer
  - NPP does not allocate memory internally (unbeknownst to the user).
  - Offer users max control and flexibility on memory management.
  - Buffer can be reused to improve performance and avoid device-memory fragmentation .



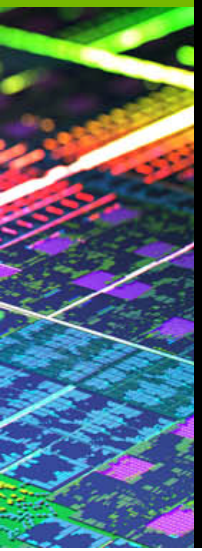
nppiMinMaxGetBufferHostSize_8u_C1R

- Parameter list:
  - NppiSize oSizeROI → oROI
  - int * hpBufferSize → &nBufferSize_Host



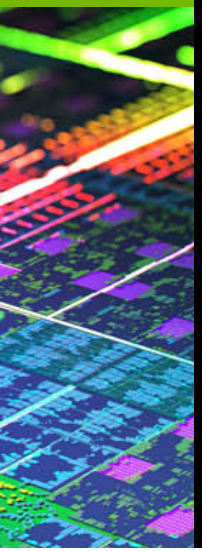
nppiMinMax_8u_C1R spec

- Parameter list:
  - const Npp8u * pSrc → pSrc_Dev
  - int nSrcStep → nSrcStep_Dev
  - NppiSize oSizeROI → oROI
  - Npp8u * pMin → pMin_Dev
  - Npp8u * pMax → pMax_Dev
  - Npp8u * pDeviceBuffer → pBuffer_Dev



Integer-Result Scaling (Part 1)

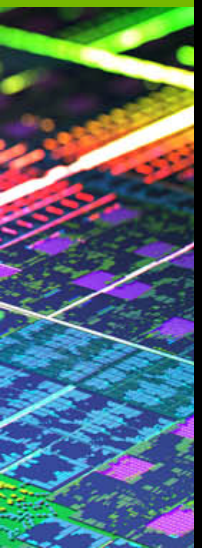
- Avoid the clamping loss on the integer data (especially on 8u and 16u images).
- Parameter "nScaleFactor" controls the amount of scaling: $\times 2^{-nScaleFactor}$
- Example: nppsSqr_8u_Sfs()
  - $255^2 = 65025$, clamped to 255.
  - Pass scale factor 8: final result = $255^2 \times 2^{-8} = 254.00390625$, which will be rounded to 254.



nppiSubC_8u_C1RSfs spec

- Shift minimum pixel value to 0.
- nppiSubC_8u_C1RSfs
  - const Npp8u * pSrc → pSrc_Dev
  - int nSrcStep → nSrcStep_Dev
  - const Npp8u nConstant → nMin, nMax, 255
  - Npp8u * pDst → pDst_Dev
  - int nDstStep → nDstStep_Dev
  - NppiSize oSizeROI → OROI
  - int nScaleFactor → 0

$$pDst(i,j) = \frac{pSrc(i,j) - nMin}{nMax - nMin} \times 255$$

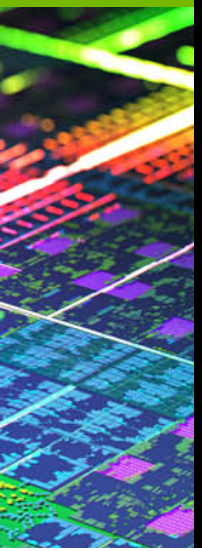


Multiplication Step

- Scale range to [0, 255].

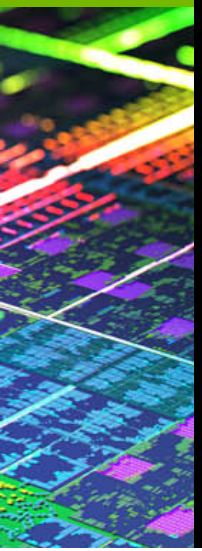
$$pDst(i,j) = (pSrc(i,j) - nMin) \times \frac{255}{nMax - nMin}$$

- Problem: nppiMulC_8u_C1IRSfs uses a single Npp8u integer as the constant multiplier, but $\frac{255}{nMax-nMin} = \frac{255}{202-92} = \frac{255}{110} \approx 2.318182$
- If multiply by 2 (nearest integer to actual value) error is 0.318182 (i.e. >30%!!!).
- Solution: Integer result scaling!



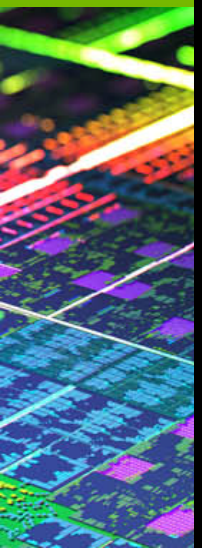
Integer-Result Scaling (Part 2)

- Reduce error using Integer-Result Scaling:
  - Express exact multiplier (2.318182) as fraction of the form $\frac{n}{2^{-s}}$
  - Where n is an integer and s is the integer-scale factor.
- With s=6 we get 2^s=64. $2.318182 \times 64 = 148.363648 \approx 148$
- $\frac{148}{64} = 2.3125$
- Difference with exact value 2.318182-2.3125 $\approx$ 0.005681 which is close to the 8-bit pixel quantization error (1/255 $\approx$ 0.003921).



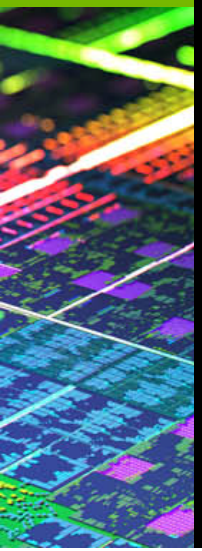
Computing optimal nScaleFactor

```
int nScaleFactor = 0;
int nPower = 1;
while(nPower * 255.0f / (nMax_Host - nMin_Host) < 255.0f)
{
    nScaleFactor ++;
    nPower *= 2;
}
```



nppiMulC_8u_C1IRSfs spec

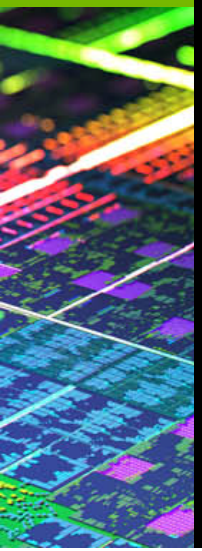
- nppiMulC_8u_C1IRSfs
  - const Npp8u nConstant → computed
  - Npp8u * pSrcDst → pDst_Dev
  - int nDstStep → nDstStep_Dev
  - NppiSize oSizeROI → oROI
  - int nScaleFactor → computed



Result

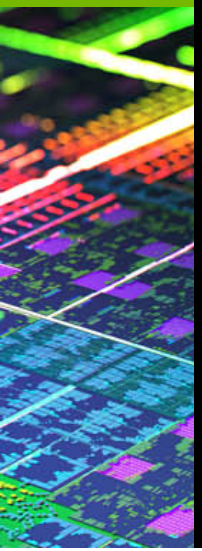
before

after



Further Reading/Resources

- NPP is freely available as part of the CUDA Toolkit at www.nvidia.com/getcuda.
- Source code samples demonstrating use of the NPP library:
  - Box Filter with NPP
  - Histogram Equalization with NPP
  - FreeImage and NPP Interoperability
  - Image Segmentation using Graphcuts with NPP



Thank you.
Q&A