aselsan

GPU Based Resolution and Contrast Enhancement for Infrared Cameras

Toygar Akgün and Murat Gevrekci
ASELSAN - Micro-electronics Guidence and Electro-Optics Division (MGEO)

1. Abstract

A massively multi-threaded CUDA implementation of spatial resolution and dynamic range enhancement technique is presented. A Bayesian technique is adopted with fixed step size in super-resolution (SR) reconstruction. A speed up of **6x** is achieved in super-resolution image formation and a speed up factor of **3x** is achieved for contrast limited adaptive histogram equalization (CLAHE). The implementation obtained by combining the multi-threaded super-resolution and CLAHE blocks runs at approximately 15 frames per second, resulting in near real-time performance. Potential improvements to further cut down the overall execution time, such as using asynchronous calls to pipeline these blocks, are currently under development.

2. Motivation

- Spatial resolution and dynamic range of imagery are both dependent on camera parameters and ambient conditions.
- Enhanced resolution is critical for tracking and recognizing small size targets with infrared imagery.

Acquire images → Pre-process → Register images → Reconstruct SR image → Contrast enh.

- Typically, once a system is deployed, it is not possible to upgrade sensor and lens systems on demand.
- Local contrast enhancement is also important since flares can severely reduce the acquired dynamic range and cripple any detection/tracking algorithms in infrared imagery.

Post-processing based resolution and contrast enhancement is required.

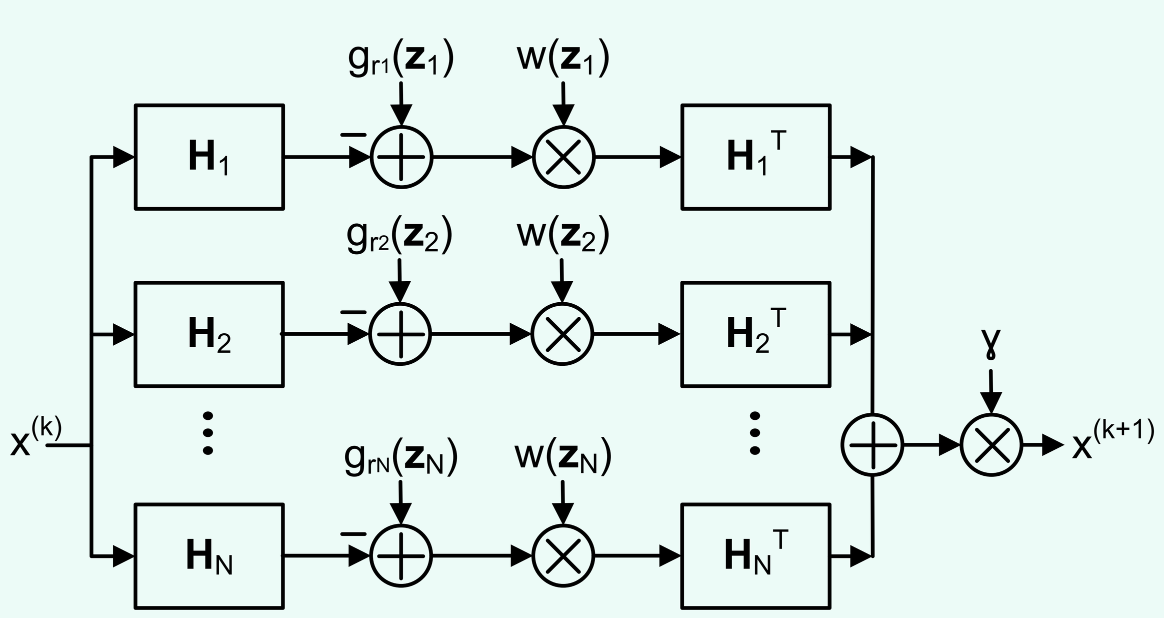
- Kepler GPU architecture provides increased perf/watt making it possible to use low-end, low-power GPUs to off-load pixel homogeneous post-processing tasks. **All results in this work are based on GT640 compared against Core i5-2400.**
- Massively multi-threaded GPU implementation can reduce the overall execution time per frame, allowing near real-time performance.
- Furthermore, reduced frame processing times allow us to increase the number of frames to be used in resolution enhancement, achieving better resolution improvement.

3. Super-resolution

Iterated back-projection method starts with an initial estimate of super resolution x(k) and modifies the super resolution estimate using forward (**H**) and backward (**H**^{T}) projections. Projection step is composed of warp, blur and decimate blocks. Contribution of residuals coming from each image is weighted by fixed step size. It is possible to use a gradient descent method for estimating the step size.

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \gamma \sum_i \mathbf{H}_i^T \mathbf{W}_i\left(g_{r_i}(\mathbf{z}_i) - \mathbf{H}_i \mathbf{x}^{(k)}\right)$$

Super resolution image formation is highly parallel as shown below:



4. Super-resolution CUDA Mapping

- Forward imaging model: **MOTION WARP** ⟶ **BLUR FILTER** ⟶ **DECIMATION**
- Backward imaging model: **UPSAMPLE** ⟶ **BLUR FILTER** ⟶ **INV MOTION WARP**

- Computational bottleneck is gradient computation. Inner most loop + heavy compute
- Bilinear upscale for initial estimate computation. Implemented by texture filtering.
- Affine motion warp is simply resampling the image on mapped grid coordinates. As such it is a very good fit for texture filtering.
- 5x5 2D blur filter is separable and implemented as two-pass global memory based convolution using shared memory caching.
- Decimation and upsampling are handled during surface reads and writes. These are augmented into the processing kernels.

BILINEAR UPSCALE	
Thread configuration	[32,32]
Register usage	17
Shared memory usage	0 B
Bank conflicts	N/A
Mem. BW efficiency (R/W)	100%
Occupancy	0,877
Time on GPU	161 us

AFFINE WARP	
Thread configuration	[32,32]
Register usage	16
Shared memory usage	0 B
Bank conflicts	N/A
Mem. BW efficiency (R/W)	100%
Occupancy	0,773
Time on GPU	245 us

SEPARABLE 2D BLUR - ROWS	
Thread configuration	[32,4]
Register usage	22
Shared memory usage	5248 B
Bank conflicts	0 %
Mem. BW efficiency (R/W)	100%
Occupancy	0,957
Time on GPU	120 us

SEPARABLE 2D BLUR - COLS	
Thread configuration	[32,8]
Register usage	32
Shared memory usage	3072 B
Bank conflicts	0 %
Mem. BW efficiency (R/W)	100%
Occupancy	0,476
Time on GPU	140 us

5. CLAHE Algorithm

Contrast **L**imited **A**daptive **H**istogram **E**qualization (CLAHE)

Acquire images
- Divide image into tiles.
- Initialize per tile processing blocks

→ **Process tiles**
- Compute local histograms.
- Clip histograms
- Create mappings for all tiles

→ **Interpolate mappings**
- For each input pixel find four neighbor mappings.
- Bi-linearly interpolate the results coming from neighboring mappings

- Locally adaptive contrast enhancement method. Adaptation is achieved by dividing the image into 32x32 blocks and extracting local histograms from each patch.
- Every pixel is mapped based on a combined map that is obtained as a bilinear interpolation of the four closest patch maps.
- Histogram clipping performed on local histograms avoids over-amplification of noise.
- Histogram equalization is not a perfect match for GPU due to random write collisions during pixel count updates and the corresponding need for atomic add.
- However, unlike the global histogram case, CLAHE is inherently local and does not require combining the locally computed histograms.
- There is also some additional parallelizable computation where an output pixel is computed as a linearly weighted version of the outputs of the closest four tiles.

6. CLAHE CUDA Mapping

- Must have 256 bin histograms for the local tile histograms. 32x32 tile size means 1024 elements max per bin.
- Histogram clipping and per tile mapping computation has three modes based on the type of distribution being used: Uniform, Exponential and Rayleigh.
- Layered 2D textures with bilinear filtering would be an ideal match to perform the final CLAHE operation. Unfortunately, the additional copy (DeviceToDevice) that is required to move mapping data to a 2D CUDA array kills the overall performance.

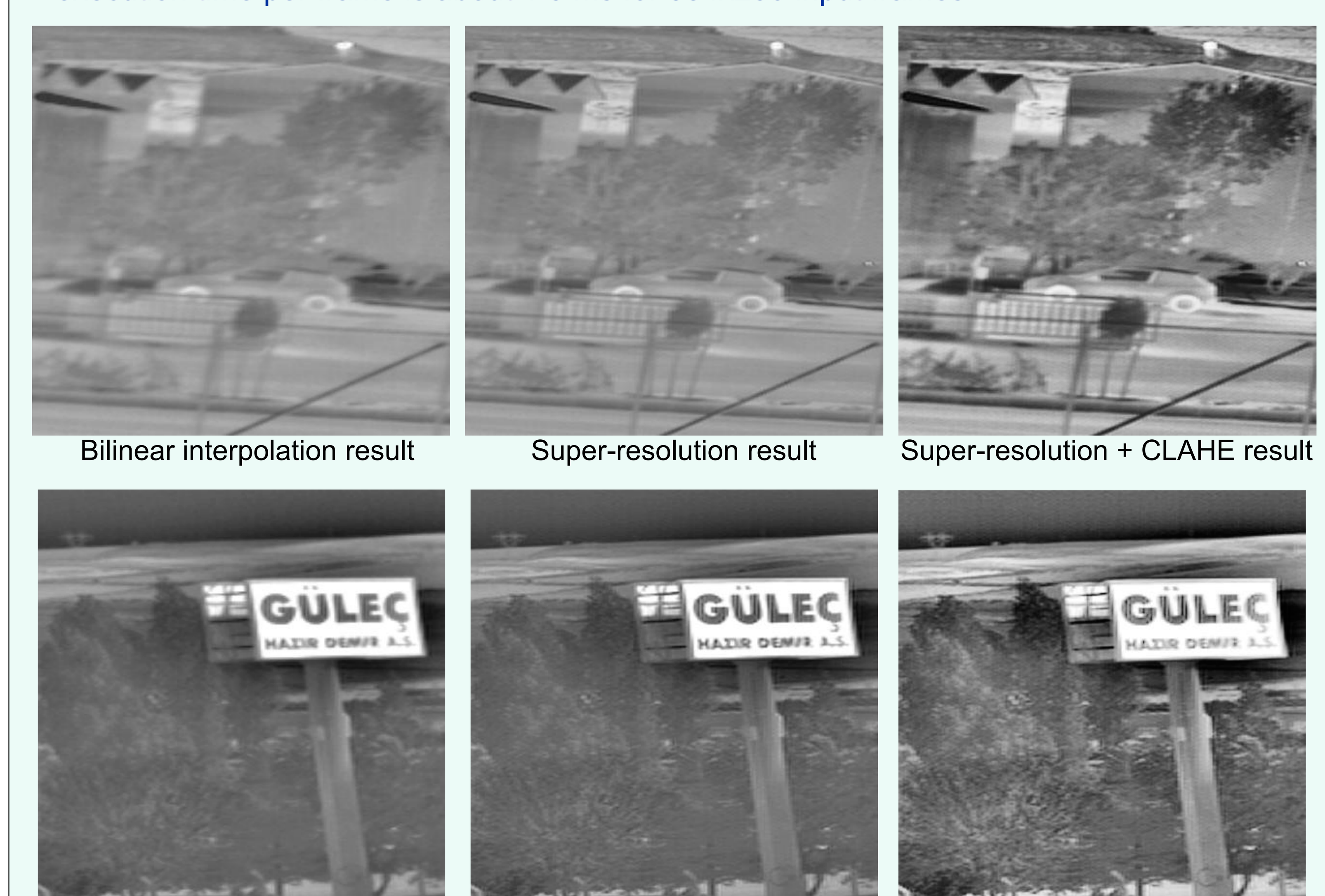
COMPUTE TILE HISTS	
Thread configuration	[32,8]
Register usage	10
Shared mem.	8KB
Bank conflicts	1.5%
Mem. BW eff. (R/W)	96/97 %
Occupancy	71.6%
Time on GPU	79 us

COMPUTE TILE MAPS	
Thread configuration	[256]
Register usage	10
Shared mem.	0 B
Bank conflicts	N/A
Mem. BW eff. (R/W)	100 %
Occupancy	87.4%
Time on GPU	28 us

EXECUTE CLAHE	
Thread configuration	[32,32]
Register usage	13
Shared mem.	0 B
Bank conflicts	N/A
Mem. BW eff. (R/W)	7/90 %
Occupancy	74.2%
Time on GPU	261 us

7. Results and Conclusions

- **GT640 @ 900 MHz with 2GB DDR3 V.S. Core i5 -2400 @ 3.1GHz with 8 GB RAM**
- For super-resolution our CUDA implementation achieved an overall speed up factor of ~6. The total execution time per frame is about 60 ms for 2X resolution enhancement factor using 5 frames of size 384x256.
- For CLAHE our CUDA implementation achieved an overall speed-up factor of ~3. The total execution time per frame is about 1.5 ms for 384x256 input frames.



Bilinear interpolation result | Super-resolution result | Super-resolution + CLAHE result

Bilinear interpolation result | Super-resolution result | Super-resolution + CLAHE result

8. References

- Murat Gevrekci, Bahadir K. Gunturk "*Super Resolution under Photometric Diversity of Images* ", *EURASIP Journal On Applied Signal Processing,* Special Issue on Super-resolution Enhancement of Digital Video, May 15 2007.
- S.M. Pizer, E.P. Amburn, J. D. Austin, et al. "*Adaptive Histogram Equalization and its Variations*", Computer Vision, Graphics, and Image Processing 39 (1987) 355-368.
- NVIDIA CUDA Programming Guide – www.nvidia.com

9. Acknowledgements

This work is supported by EU-CIG project "POCS Based Depth Super-resolution" with grant number 267107.