Implementations Impact on Iterative Image Processing for Embedded GPU

Thomas Romera [1, 2], Andrea Petreto [1,2], Florian Lemaitre [1], Manuel Bouyer [1], Quentin Meunier [1], Lionel Lacassagne [1]

Sorbonne Université, CNRS, LIP6 [1]

LHERITIER - ALCEN [2]

EUSIPCO 2021 - 24 august

Objectives

Main objective:

To achieve the fastest implementation of the TV-L1 optical flow algorithm on embedded systems

- Focus on embedded GPU implementation:
  - Higher parallel processing power (vs. CPU)
  - Lower energy consumption (vs. discrete GPU)
- 3 NVIDIA platforms considered [7]:

Board	Process	CPU	Fmax (GHz)	GPU	Fmax (GHz)	Power max (Watt)
AGX	12 nm	8×Carmel	2.27	512 C Volta	1.4	30
TX2	16 nm	4×A57 + 2×Denver 2	2.00	256 C Pascal	1.3	20
Nano	12 nm	4×A57	1.43	128 C Maxwell	0.9	10

Table 1: NVIDIA Jetson Systems Technical Specifications.

TV-L1 Optical Flow Estimation

- Robustness to flow discontinuities, brightness variations, occlusions and noise [12]

  $\rightarrow$ Good compromises between all those properties
- Used in many recent video processing applications (video denoising [8], action recognition [4], 3D scene reconstruction [5])
- Used as a basis for more complex optical flow estimations [11, 1, 10]
- Pyramidal multi-scale iterative stencil algorithm
- Iterative parameters: warps per scale and numerical iterations per warp

$\Rightarrow$ Study of mono-scale and mono-warp versions

- Simpler analysis
- Fixed interpolation type, scaling method or warping number

TV-L1 GPU Implementation

- NVIDIA CUDA C++ API [6] implementation
- Performed optimisations:
  - Operator fusion [3, 9]: less function launch overhead, decreased memory accesses
  - Shared memory: small fast memory shared by all threads of a thread-block
  - No warp divergence: no sequential execution of conditional code branches
  - *float2* usage: two 32-bit floating point numbers packed in the same 64-bit type.
    → Increased memory bandwidth due to vectorised loads/stores
  - *half2* usage: two 16-bit floating point numbers packed in the same 32-bit type.
    → Same *float2* benefits and added subword parallelism
  - Thread-block size tuning: optimal block size tested and used
- Comparisons:
  - Use of 32 bits floating point numbers (F32) using *float* and *float2* type
  - Use of 16 bits floating point numbers (F16) using *half* and *half2* type
  - Comparison with optimised SIMD ARM Neon versions (F32 and F16)
  - Comparison with OpenCV CPU and GPU reference implementations
  - *"F32/F16 Best GPU"* versions include all listed optimisations

Execution Time Comparison

CPU vs. GPU, F32 vs. F16, and impact of optimisations

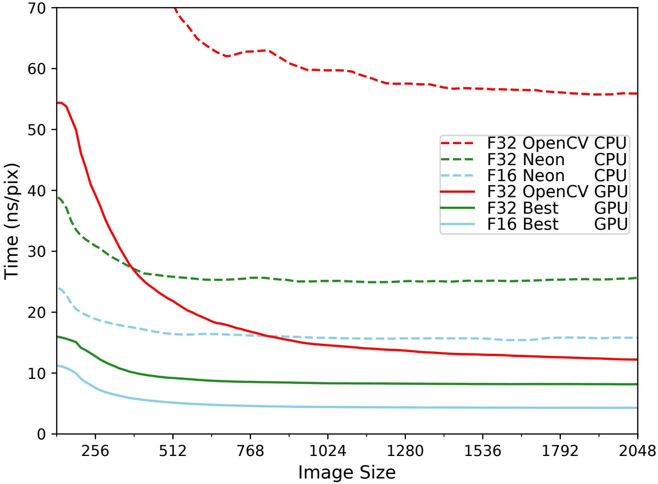

Figure 1: TV-L1 CPU/GPU 1 scale, 1 warp, 10 iterations on Jetson AGX.

GPU speedup vs. CPU:

- **OpenCV**: $\times 4.7$
- **F32**: $\times 3.3$
- **F16**: $\times 4.0$

$\Rightarrow$ Similar on all boards

$\Rightarrow$ *F32 Neon CPU* faster than *F32 OpenCV GPU* for images $< 400 \times 400$ pixels

$\Rightarrow$ *F16 Neon CPU* for images $< 800 \times 800$ pixels

$\Rightarrow$ Optimised *F32 Best GPU* and *F16 Best GPU* always faster than CPU

Performance and Energy Efficiency

CPU vs. GPU, F32 vs. F16, and impact of optimisations

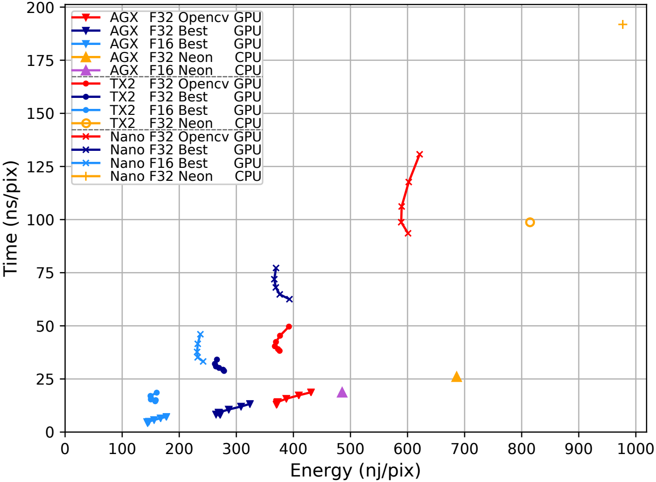

Figure 2: Time (ns/pix) and energy (nJ/pix) operating points for CPU and GPU TV-L1 implementations on each Jetson boards for several clock frequencies.

CPU/GPU energy ratio:

- **F32**: $\times 2.6$ (AGX)
- **F16**: $\times 3.4$ (AGX)

⇒ GPU is faster and consumes less energy than CPU

⇒ AGX is faster and consumes less energy than the other Jetson systems

Conclusion

Image Resolution	CPU time (ms)		GPU time (ms)		
	F32 Neon	F16 Neon	OpenCV	F32 Best	F16 Best
4K UHD (3840×2160)	214.4	132.8	96.0	62.5	**34.4**
FHD (1920×1080)	52.8	52.8	27.3	13.7	**8.6**
FHD/4 (960×540)	13.1	8.5	9.2	4.0	**2.5**

Table 2: TV-L1 GPU 1 scale, 1 warp, 10 iterations on Jetson AGX.

→ 10 iterations of TV-L1 mono-scale on 4K images in less than 40 ms

→ Image size impact on processing time:

- F16 Best FHD: 4.1 ns/pix (8.6 ms / 3840×2160 pixels)
- F16 Best 4K UHD: 4.1 ns/pix (34.4 ms / 1920×1080 pixels)

⇒ Testing the scalability of our optimisations on larger images and GPUs for more demanding applications

Thank you

Bibliography I

[1] Coloma Ballester et al. "A TV-L1 optical flow method with occlusion detection". In: *Proceedings of the 2012 Joint DAGM (German Association for Pattern Recognition) and OAGM Symposium*. 2012, pp. 31–40.

[2] Linchao Bao et al. "A comparison of TV-L1 optical flow solvers on GPU". In: *GTC Posters* 6 (2014).

[3] Lionel Lacassagne et al. "High level transforms for SIMD and low-level computer vision algorithms". In: *Proceedings of the 2014 Workshop on Programming models for SIMD/Vector processing (WPMVP)*. 2014, pp. 49–56.

[4] Ji Lin, Chuang Gan, and Song Han. "Tsm: Temporal shift module for efficient video understanding". In: *Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 7083–7093.

[5] Richard A Newcombe and Andrew J Davison. "Live dense reconstruction with a single moving camera". In: *Proceedings of the 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. 2010, pp. 1498–1505.

Bibliography II

[6] NVIDIA Corp. *CUDA C++ Programming Guide Version 11.2.0*. url = https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html. Jan. 2021.

[7] NVIDIA Corp. *Jetson Portfolio*. url = https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/. Feb. 2021.

[8] Andrea Petreto et al. "A New Real-Time Embedded Video Denoising Algorithm". In: *Proceedings of the 2019 Conference on Design and Architectures for Signal and Image Processing (DASIP)*. 2019, pp. 47–52.

[9] Andrea Petreto et al. "Energy and execution time comparison of optical flow algorithms on SIMD and GPU architectures". In: *Proceedings of the 2018 Conference on Design and Architectures for Signal and Image Processing (DASIP)*. 2018, pp. 25–30.

[10] René Ranftl, Kristian Bredies, and Thomas Pock. "Non-local total generalized variation for optical flow estimation". In: *Proceedings of the 2014 European Conference on Computer Vision (ECCV)*. 2014, pp. 439–454.

Bibliography III

[11] Andreas Wedel et al. "An improved algorithm for TV-L1 optical flow". In: *Statistical and geometrical approaches to visual motion analysis*. Springer, 2009, pp. 23–45.

[12] Christopher Zach, Thomas Pock, and Horst Bischof. "A duality based approach for realtime TV-L1 optical flow". In: *Proceedings of the 29th DAGM Conference on Pattern Recognition (DAGM GCPR)*. 2007, pp. 214–223.

TV-L1

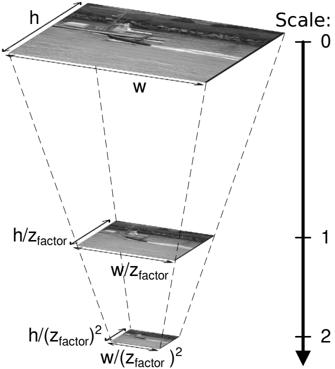

Figure 3: Pyramidal TV-L1 structure

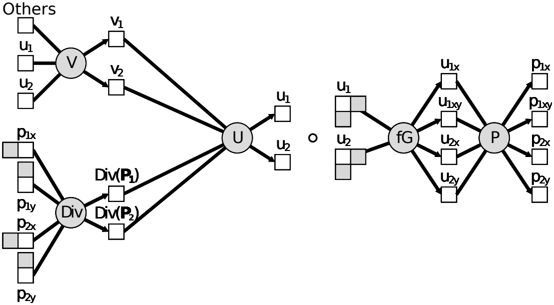

Figure 4: TV-L1 iteration without algorithmic fusion

TV-L1 Execution Time

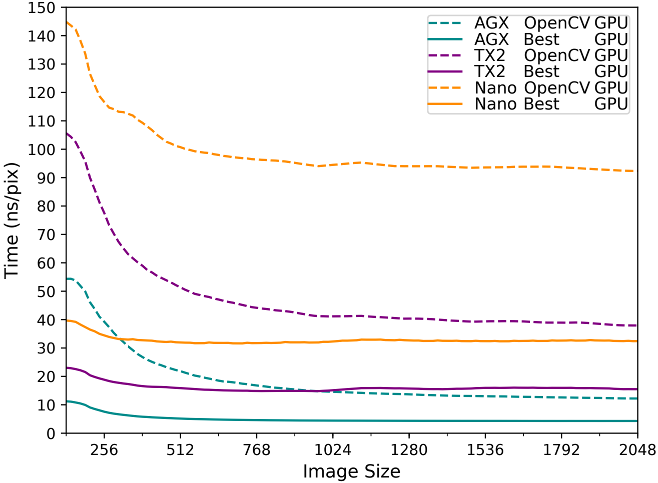

Figure 5: TV-L1 execution time (ns/pix) comparison with OpenCV on the 3 Jetson boards.

F32 vs. F16 Execution Time

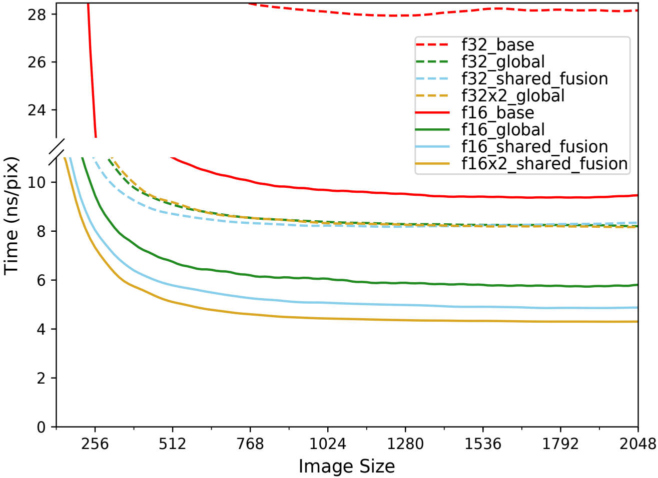

Figure 6: F32 and F16 execution time (ns/pix) on AGX.

State-of-the-Art Execution Time Comparison

Algorithm	Warps	Normalised GPU cycles	Speedup vs F_{32}	Speedup vs F_{16}
Our F16 Best GPU	1	1.6	2	-
Our F32 Best GPU	1	3.2	-	-
[2] TV-L1 (DL solver)	1	12.9	7.9	15.8
[12] TV-L1 introduction	1	219.8	143.7	287.4
Our F16 Best GPU	25	46.0	1.9	-
Our F32 Best GPU	25	86.6	-	-
[11] P(GPU)	25	156.2	1.8	3.4

Table 3: Execution time comparison between State-of-the-Art TV-L1 implementation and our fastest versions.