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]
[1]*Sorbonne Université, CNRS, LIP6, F-75005 Paris, France*
[2]*LHERITIER - ALCEN, F-95862, Cergy-Pontoise, France*
{firstname.lastname}@lip6.fr

***Abstract*—The emergence of low-power embedded Graphical Processing Units (GPUs) with high computation capabilities has enabled the integration of image processing chains in a wide variety of embedded systems. Various optimisation techniques are however needed in order to get the most out of an embedded GPU. This paper explores several optimisation methods for iterative stencil-like image processing algorithms on embedded NVIDIA GPUs using the Compute Unified Device Architecture (CUDA) API. We chose to focus our architectural optimisations on the TV-L1 algorithm, an optical flow estimation method based on total variation (TV) regularisation and the L1 norm. It is widely used as a model for more complex optical flow estimations and is used in many recent video processing applications. In this work we evaluate the impact of architecture-oriented optimisations on both execution time and energy consumption on several Nvidia Jetson GPU embedded boards. Results show a speedup up to $3\times$ compared to State-of-the-Art versions as well as a $2.6\times$ decrease in energy consumption.**

***Index Terms*—GPU, Embedded System, Image Processing, TV-L1, Optical flow, Energy Consumption**

I. Introduction

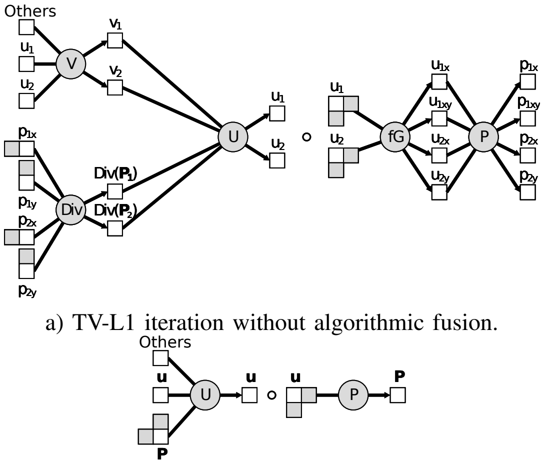
Today's embedded image processing chains are becoming more and more complex and demand a lot of computing capabilities. Design constraints such as real-time processing and limiting power consumption are increasingly hard to achieve. Using embedded GPUs significantly increases the parallel processing power of the embedded system. The Nvidia Jetson boards (TX2, AGX Xavier and Nano) [1] are examples of the latest embedded GPUs architectures. They are a family of embedded computing boards carrying a Tegra SoC that includes an ARM CPU and an Nvidia GPU, ready to be programmed using the CUDA programming model.

Even if such systems allow for heavier embedded image processing, the most complex algorithms still need trade-offs and optimisations in order to both decrease processing time and energy consumption. This papers explores such optimisations and trade-offs on iterative stencil-like image processing algorithms, specifically on the TV-L1 optical flow estimation.

Optical flow represents the apparent motion of objects, surfaces and edges in a visual scene [2]. It is a major tool in video enhancement [3], [4] and computer vision [5], [6]. First pioneered by Horn and Schunck [7] and by Lucas and Kanade [8] in 1981, hundreds of different algorithms now exist. Most of them are indexed and evaluated in the Middlebury database [9]. These various methods differ by their computing speed and accuracy. The most qualitative methods are usually also the slowest ones.

The TV-L1 algorithm is a good candidate for optimisations on embedded systems. First introduced by Zach, Pock and Bischof in 2007, the TV-L1 optical flow is an iterative numerical scheme based on the minimisation of an energy function [10]. It is a stencil-like iterative algorithm using various intermediate data arrays, which is well suited for algorithmic optimisations. Unlike other simpler methods such as Horn and Schunck's, TV-L1 allows for discontinuities in the optical flow. Furthermore, it is widely used as a model for more complex optical flow estimations [11]–[13] and is used in many recent video processing applications such as video denoising [3], action recognition [14]–[16] or 3D scene reconstruction [17]. Different implementations exist on CPU namely the original [10] and improved [11] versions, a parallel OpenMP version [18] and a SIMD version [19]. FPGA implementations have also been developed [20] and have been optimised in memory allocation and power consumption [21]. Finally, GPU implementations have been developed for the original [10], improved [11] and further optimised TV-L1 versions [22], [23].

Many newer optical flow methods have been introduced in the last few years based on deep-learning [24]–[26]. Those methods are not good candidates for embedded systems since they are slower and require large if not multiple GPUs to run and thus have a large power requirement (typically several hundred watts). Even the fastest method to our knowledge, FlowNet 2.0 [24], takes 7 ms to process 1024×436 pixel images on an Nvidia GTX 1080. This GPU contains $5\times$ the number of CUDA cores and has a clock frequency 25% higher than the most powerful embedded platform tested in this work: the Jetson AGX Xavier. If we estimate the time needed to run FlowNet 2.0 on an AGX board by scaling the number of cores and the frequency, we find that FlowNet 2.0 runs at 98 ns/pix. In comparison, the TV-L1 OpenCV implementation tested in our paper runs at 20.1 ns/pix. Our goal is to be able to process frames of 2048×2048 pixels at 25 images per seconds (a processing time of 40 ms), corresponding to a processing time of 9.54 ns/pix. In this context, the deep-learning algorithm requires an acceleration of 10 times while the TV-L1 algorithm requires an acceleration of 2 times. TV-L1 is therefore a much better candidate for embedded applications than deep-learning



a) TV-L1 iteration without algorithmic fusion.

b) Optimised TV-L1 iteration with fusion.

Figure 1: Consumer/Producer representation of a TV-L1 iteration. The white squares corresponds to the central pixels and the grey squares to the left, right, upper or lower pixels.

methods.

In this work we introduce several optimised GPU implementations and evaluate both time and energy consumption on embedded systems. We also compare our new implementations to other State-of-the-Art versions.

II. CUDA GPU Optimisations

As shown in [19], figure 1 illustrates a data consumer/producer representation of the main steps of one iteration of the TV-L1 algorithm. Part a) represents the 5 steps:

- **V** to estimate the relaxation term
- **Div** to compute the divergence
- **U** to update the optical flow estimation $U = (u_1, u_2)$
- **fG** to compute the forward gradient
- **P** to update the double vector field $P = (p_1, p_2) = (p_{1x}, p_{1y}, p_{2x}, p_{2y})$

The divergence is estimated with backward differences and the forward gradient with forward differences [18]. A pyramidal approach is used to compute displacements larger than 1 pixel. A warping step between each scale is used to propagate the solution across pyramid levels. Warps consist in a movement compensation using interpolation (in our case bicubic). Additional warps can be performed on each level to get more robust and accurate results.

The main algorithmic transformation used in this work for GPU implementations is operator fusion [19], [27]. This reduces the number of memory accesses, improves data locality and improves arithmetic intensity. This also reduces the total kernel launch overhead due to many kernel calls (between 2 and 5 μs per kernel call). Part b) of the figure 1 shows the 2 main steps of the TV-L1 algorithm after fusion: one step to update U and one step to update P. This representation also illustrates the difficulty to further regroup those two remaining steps into one due to data dependencies.

Those data dependencies also make pipelining difficult even after algorithmic fusion. The main advantage of iterations pipeline is to improve the data reuse within fast local memory. On GPU this can be done by using the *shared memory*. This memory is faster than the global memory and is accessible by all the threads of a thread block. However, on the embedded GPU we tested, we were not able to pipeline more than 2 iterations. Because of the dependencies, an apron proportional to the pipeline depth is needed. There is not enough shared memory available to fit all the necessary data to pipeline more than 2 iteration of TV-L1. The Jetson AGX Xavier uses the latest Nvidia GPU architecture available on embedded systems, the Volta architecture, which only allows up to 48 KB of shared-memory per thread block. For a 3-iteration pipeline, we would need 12 intermediate 7x7 buffers which would require 75 KB of shared memory using 32-bit floating point numbers. There is not enough shared memory per block on GPU to launch such a kernel. Moreover, no newer Nvidia architecture has been announced for embedded systems yet. The AGX platform will last a long time before being obsolete. With only 2 iterations pipelined, there is not enough data reuse and the computation is slower than without pipeline.

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

Table 1: Technical specifications of the NVIDIA Jetson boards.

We present two kinds of TV-L1 optimisation: global and shared_fusion.

global implements the algorithmic fusion shown in figure 1b using only the global GPU memory. shared_fusion computes both U and P in one step using the shared memory. All the elements, including a border outside of the thread block, are loaded in shared memory for intermediate data reuse.

global and shared_fusion are declined in both 32-bit (F_{32}) and 16-bit (F_{16}) floating point versions. As shown in [28] the use of F_{16} is sufficient for optical flow. Furthermore, the original implementation in [10] already uses F_{16} for parts of the algorithm on GPU. We also used CUDA vector types *__float2* (resp. *__half2*) in our f32x2 (resp. f16x2) implementations. The *__half2* type allows us to use subword parallelism not available with *__float2* type.

III. Experimental Evaluation

A. Evaluation Benchmark

For our experimentation we used three Nvidia Jetson embedded platforms detailed in table 1. Three kinds of evaluation are performed. We first compare the execution time of CPU and GPU implementations on the three considered systems. We measure the execution time of a mono-scale configuration with 1 bicubic warping step and 10 iterations. We use square images of size from 128×128 up to 2048×2048 pixels and the computation time is normalised in nano-seconds per pixel (ns/pix).

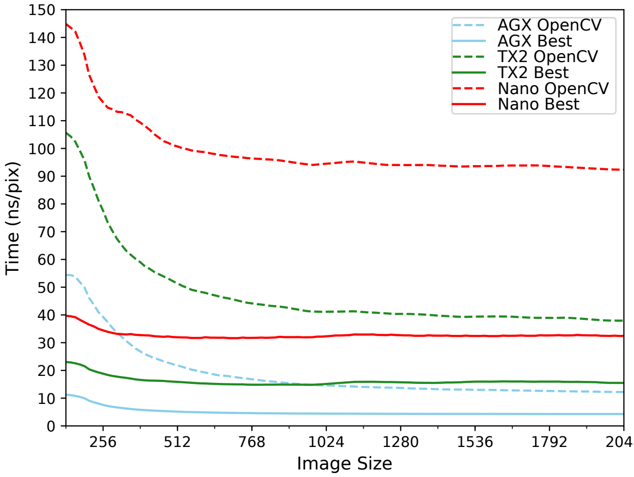


Figure 2: TV-L1 execution time (ns/pix) comparison with OpenCV on CPU. Our best implementation here corresponds to the f16x2_shared_fusion version on GPU.

Next, we evaluate the power consumption of the same implementations by means of an external electronic board developed to this effect. We vary the GPU clock from the lowest to the highest frequency and record both the time and the energy taken to compute 1 warp and 10 iterations on images of size 2048 × 2048 pixels. The energy results are also normalised in nano-joules per pixel (nJ/pix). We explore the (time per pixel, energy per pixel) space to find the fastest and most energy efficient implementations and platforms.

Finally, we compare our work to State-of-the-Art TV-L1 GPU implementations.

B. Timing Result Analysis

Figure 2 shows the processing time in ns/pix depending on the image size on each platform. Our fastest version called Best is compared to the open source GPU implementation of TV-L1 provided by OpenCV. Overall the results are similar on each boards with a speedup of 2.8× on the AGX, 2.5× on the TX2 and 2.9× on the Nano. As such, we focus the rest of our analysis on the AGX which possesses the newest GPU architecture (Volta), the biggest and fastest GPU (512 cores, 1.3 GHz) out of the three tested Jetson boards.

Figure 3 shows the processing speed of different implementations of TV-L1 on the AGX board, namely:

- OpenCV XX: F_{32} OpenCV implementation on the XX architecture (CPU or GPU)
- XX_neon CPU: Optimised Neon SIMD CPU version using XX (F_{32} or F_{16}) computation format,
- XX_base: GPU baseline version using XX format
- XX_global: GPU optimised implementation with operator fusion using only the global memory and XX format
- XX_shared_fusion: GPU optimised implementation with operator fusion using shared memory and XX format

The timing results in figure 3 shows that the optimised CPU Neon versions are faster than the f32_base GPU version. The f32_neon CPU version is also faster than the OpenCV GPU version for images less than 400 × 400 pixels and the f16_neon CPU version is faster for images less than 800 × 800. Our f16x2_shared_fusion GPU version is 3× faster than OpenCV GPU and 7× faster than f32_base. Better data reuse, less reads and writes to global memory, along with smaller data size due to the use of F_{16} lead to a lower memory footprint of our implementations. Table 2 shows this increase in the effective memory bandwidth.

Figure 4 shows a more detailed comparison between GPU implementations. We can see that all the F_{32} versions ultimately

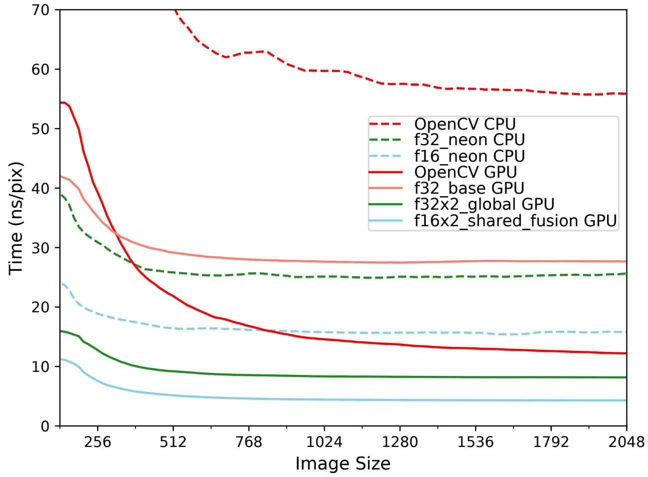


Figure 3: Execution time (ns/pix) of TV-L1 implementations on AGX.

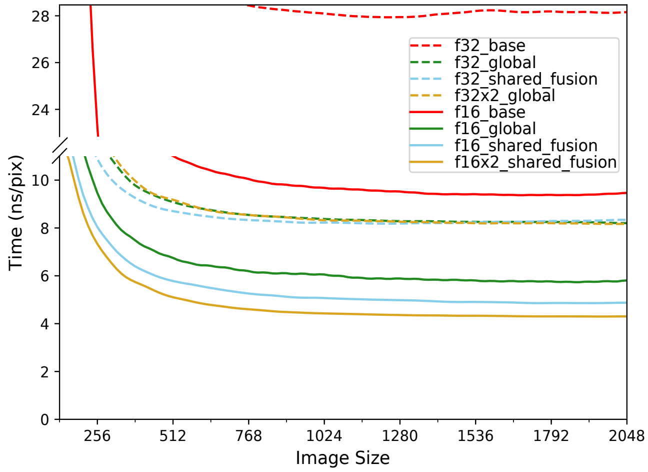


Figure 4: F_{32} and F_{16} execution time (ns/pix) on AGX.

Version	Time (ns/pix)	Bandwidth (GB/s)	Throughput (GFLOP/s)
OpenCV CPU (F_{32})	56	40.0	14.6
f32_neon CPU	26	51.3	24.3
f16_neon CPU	16	41.7	39.4
OpenCV GPU (F_{32})	12	125.2	51.2
f32x2_shared_fusion GPU	8	178.1	79.7
f16x2_shared_fusion GPU	4	243.2	119.3

Table 2: Execution time, memory bandwidth and computational throughput of several TV-L1 implementations on the Jetson AGX.

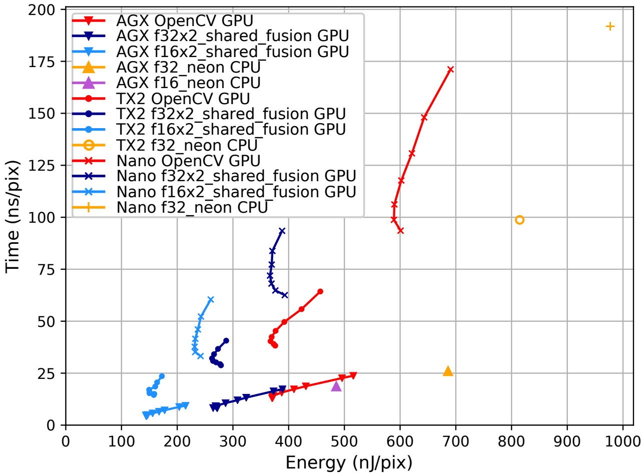


Figure 5: Time (ns/pix) and energy (nJ/pix) operating points for several TV-L1 implementations on each Jetson boards. Several clock frequencies are tested for each versions.

plateau at 8.4 ns/pix. In F_{32}, the use of shared memory or f32x2 does not provide any speedup. In F_{16} however, f16_shared_fusion (4.9 ns/pix) performs better than f16_global (5.6 ns/pix) thanks to more reuse of data in shared memory. More data can be allocated in shared memory since F_{16} are twice as small as F_{32}. Furthermore, the f16x2_shared_fusion is even faster at 4.3 ns/pix since f16x2 enables us to use subword parallelism which increases computational throughput as shown in table 2. This parallelism is not supported in F_{32}.

C. Power Consumption Result Analysis

Figure 5 presents the results in the (time per pixel, energy per pixel) space for our TV-L1 implementations on GPU along with the optimised Neon CPU versions and the GPU OpenCV version. Each point along each curve corresponds to a different clock frequency. Only the maximum frequency is shown here for each Neon CPU versions. The AGX achieves the best performance in both terms of runtime and energy consumption. The TX2 board is a close second in terms of energy consumption. The fastest GPU implementation is f16x2_shared_fusion and is 4× faster and 3.4× more power efficient than the best CPU version (f16_neon) on the AGX. It is also 3× faster and 2.6× more energy efficient than OpenCV. The f16x2_shared_fusion version is both the fastest and the most power efficient implementation on every platform.

The figure 6 provides more detailed results for the AGX on 2048×2048 pixels image at various GPU frequencies. Figure 6a, shows the board's total energy consumption in nJ/pix. In figure 6b, the system's idle energy consumption is subtracted from the total energy. Figure 6c shows the computation time in CPP. There is no significant difference in energy consumption between the frequency minimising energy consumption and the highest frequency for the f16x2_shared_fusion version.

We can see that for f16x2_shared_fusion the energy consumption is almost constant for all frequencies. As shown in c, this can be explained by an almost linear computation speedup compared to frequency augmentation (the CPP are constant). This is not the case with the other implementations where at some point the CPP and the energy consumption start to increase. This change appears when the GPU frequency grows too high compared to the memory frequency. From this point on the memory bandwidth becomes a bottleneck. This behaviour remains valid for all 3 platforms.

Algorithm	Warps	Normalised GPU cycles	Speedup vs F_{32}	Speedup vs F_{16}
Our f16x2_shared_fusion	1	1.6	2	-
Our f32x2_shared_fusion	1	3.2	-	-
[23] TV-L1 (DL solver)	1	12.9	7.9	15.8
[10] TV-L1 introduction	1	219.8	143.7	287.4
Our f16x2_shared_fusion	25	46.0	1.9	-
Our f32x2_shared_fusion	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.

D. State-of-the-Art Comparison

In this section we compare our implementations to other State-of-the-Art ones. Since source codes are no longer available, we were not able to perform a real comparison on identical platforms. Thus, the execution time for each algorithm is normalised according to GPU hardware and the algorithm configuration with the following formula:

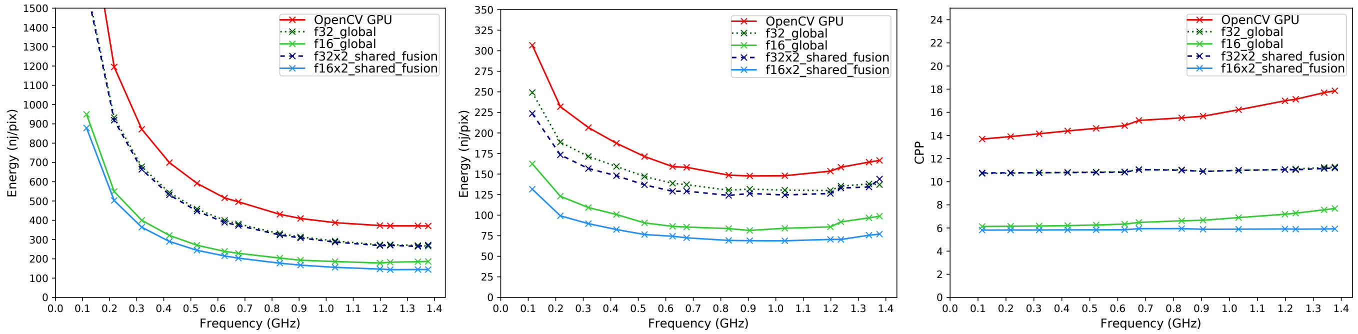
$$C = \frac{T \times F}{N_{cores} \times N_{pix} \times N_{iter}}$$

where C is the number of cycles per core per iteration, T is the time of a given benchmark, F is the GPU frequency, N_{cores} the number of GPU cores, N_{pix} the number of pixel in the image and N_{iter} the number of iterations.

Table 3 presents the execution time comparison between State-of-the-Art and f16x2_shared_fusion implementations. Equivalent configurations and parameters (i.e. same number of scales, warps per scales and iterations per warps and same optical flow dataset) were used whenever possible. The first part of the table corresponds to the "light" TV-L1 configuration found in other publications with 3 scales, 1 warp per scale and 50 iterations per warp. Here, the biggest acceleration is with [10] with a speedup of 143.7×. We should note that this version uses both an older GPU architecture and language (Cg). For the same algorithm on close GPU architectures, we observe a runtime acceleration of 7.9×. The second part of the table corresponds to a heavier and slower configurations with 5 scales, 25 warps per scale and 10 iterations per warp yielding more accurate results. Compared to the other implementations, a speed-up of 4.4× is achieved.

IV. Conclusion

In this paper, we show several optimisation technique for the TV-L1 optical flow iterative algorithm on embedded GPUs. Thanks to more data reuse, using the shared memory and using F_{16} and sub-word parallelism, we can achieve 3× speedup as well as a 2.6× decrease in energy consumption, compared to other State-of-the-Art GPU implementations. On the newest AGX board, our best embedded GPU implementation is 4× faster and consumes 2.6× less energy than our fastest CPU



a) Total energy consumption at different frequencies

b) Energy consumption without idle energy at different frequencies

c) Cycles per pixel at different frequencies

Figure 6: Energy (nJ/pix) with/without idle consumption and time (Cycles Per Pixel: CPP) on the AGX board for 2048 × 2048 pixels images.

SIMD implementation. Real-time processing at 25 frames per second for images up to 2048 × 2048 pixels can be achieved on GPU while lowering energy consumption. These optimisations can be applied to any optical flow algorithms and any iterative stencil-like algorithm.

Acknowledgement

This work has been partially funded by the *Direction générale de l'armement* (DGA), french Ministry of Armed Forces.

References

[1] NVIDIA Corp., "Jetson portfolio," February 2021, url = https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/.

[2] A. Burton and J. Radford, *Thinking in Perspective: Critical Essays in the Study of Thought Processes*, ser. Psychology in progress. Methuen, 1978. [Online]. Available: https://books.google.fr/books?id=CSgOAAAAQAAJ

[3] A. Petreto, T. Romera, F. Lemaitre, I. Masliah, B. Gaillard, M. Bouyer, Q. L. Meunier, and L. Lacassagne, "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.

[4] T. Xue, B. Chen, J. Wu, D. Wei, and W. T. Freeman, "Video enhancement with task-oriented flow," *International Journal of Computer Vision (IJCV)*, vol. 127, no. 8, pp. 1106–1125, 2019.

[5] Z. Kalal, K. Mikolajczyk, and J. Matas, "Tracking-learning-detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 34, no. 7, pp. 1409–1422, 2011.

[6] J. D. Adarve and R. Mahony, "A filter formulation for computing real time optical flow," *IEEE Journal of Robotics and Automation Letters (RA-L)*, vol. 1, no. 2, pp. 1192–1199, 2016.

[7] B. K. Horn and B. G. Schunck, "Determining optical flow," *Journal of Artificial Intelligence (AIJ)*, vol. 17, no. 1-3, pp. 185–203, 1981.

[8] B. D. Lucas, T. Kanade *et al.*, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI)*, 1981.

[9] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, and R. Szeliski, "A database and evaluation methodology for optical flow," *International Journal of Computer Vision (IJCV)*, vol. 92, no. 1, pp. 1–31, 2011.

[10] C. Zach, T. Pock, and H. 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.

[11] A. Wedel, T. Pock, C. Zach, H. Bischof, and D. Cremers, "An improved algorithm for TV-L1 optical flow," in *Statistical and geometrical approaches to visual motion analysis*. Springer, 2009, pp. 23–45.

[12] C. Ballester, L. Garrido, V. Lazcano, and V. Caselles, "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.

[13] R. Ranftl, K. Bredies, and T. 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.

[14] J. Carreira and A. Zisserman, "Quo vadis, action recognition? a new model and the kinetics dataset," in *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6299–6308.

[15] C. Feichtenhofer, A. Pinz, and R. P. Wildes, "Spatiotemporal multiplier networks for video action recognition," in *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 4768–4777.

[16] J. Lin, C. Gan, and S. 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.

[17] R. A. Newcombe and A. 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.

[18] J. S. Pérez, E. Meinhardt-Llopis, and G. Facciolo, "TV-L1 optical flow estimation," *Image Processing On Line (IPOL)*, vol. 2013, pp. 137–150, 2013.

[19] A. Petreto, A. Hennequin, T. Koehler, T. Romera, Y. Fargeix, B. Gaillard, M. Bouyer, Q. L. Meunier, and L. Lacassagne, "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.

[20] I. Beretta, V. Rana, A. Akin, A. A. Nacci, D. Sciuto, and D. Atienza, "Parallelizing the chambolle algorithm for performance-optimized mapping on FPGA devices," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 15, no. 3, pp. 1–27, 2016.

[21] P. Garcia, D. Bhowmik, R. Stewart, G. Michaelson, and A. Wallace, "Optimized memory allocation and power minimization for FPGA-based image processing," *Journal of Imaging*, vol. 5, no. 1, p. 7, 2019.

[22] E. d'Angelo, J. Paratte, G. Puy, and P. Vandergheynst, "Fast TV-L1 optical flow for interactivity," in *Proceedings of the 18th IEEE International Conference on Image Processing (ICIP)*, 2011, pp. 1885–1888.

[23] L. Bao, H. Jin, B. Kim, and Q. Yang, "A comparison of TV-L1 optical flow solvers on GPU," *GTC Posters*, vol. 6, 2014.

[24] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," in *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2462–2470.

[25] G. Yang and D. Ramanan, "Volumetric correspondence networks for optical flow," *NeurIPS*, vol. 5, p. 12, 2019.

[26] T.-W. Hui and C. C. Loy, "Liteflownet3: Resolving correspondence ambiguity for more accurate optical flow estimation," in *Proceedings of the 2020 European Conference on Computer Vision (ECCV)*, 2020, pp. 169–184.

[27] L. Lacassagne, D. Etiemble, A. Hassan Zahraee, A. Dominguez, and P. Vezolle, "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.

[28] S. Piskorski, L. Lacassagne, S. Bouaziz, and D. Etiemble, "Customizing cpu instructions for embedded vision systems," in *Proceedings of the 2006 International Workshop on Computer Architecture for Machine Perception and Sensing (CAMPS)*, 2006, pp. 59–64.