Energy and Execution Time Comparison of Optical Flow Algorithms on SIMD and GPU Architectures

Andrea Petreto[1][2], Arthur Hennequin[1], Thomas Koehler[1], Thomas Romera[1], Yohan Fargeix[1], Boris Gaillard[2], Manuel Bouyer[1], Quentin L. Meunier[1] and Lionel Lacassagne[1]
[1] Sorbonne University, CNRS, LIP6, F-75005 Paris, France. Email: firstname.lastname@lip6.fr
[2]Lhéritier - Alcen, F-95862, Cergy-Pontoise, France. Email: bgaillard@lheritier-alcen.com

***Abstract*—This article presents and compares optimized implementations of two optical flow algorithms on several target boards comprising multi-core SIMD processors and GPUs. The two algorithms are Horn-Schunck (HS) and TV-L1, and have been chosen because they are both well-known, and because of their different computational complexity and accuracy. For both algorithms, we have made parallel optimized SIMD implementations, while HS has also been implemented on GPUs. For each algorithm, the comparison between the different versions and target boards is carried out in a two-dimensional fashion: in terms of computing speed – in order to achieve real-time computation – and in terms of energy consumption since we target embedded systems. The results show that for HS, the GPUs are the most efficient in both dimensions, able to process in real-time performances (25 frames per second) up to 8Mpix images for 0.35J per image, against 1.8Mpix images for 0.24J per image on CPU. The results also highlight the impact of optimizations on TV-L1: far slower than HS without optimization, it can almost match its performance after optimization on CPU, and can achieve real-time performances with 0.25J for 1.4Mpix images. We hope these results will help developers design optical flow embedded systems.**

I. Introduction

Embedded systems must satisfy the antagonist constraints of real-time processing and power consumption, especially for computer vision. Optical flow is a family of algorithms which are used to estimate the apparent velocity of every point in a pair of images. These algorithms are used in a wide range of applications, from movement compensation on cameras to autonomous vehicles.

To estimate optical flow in real time is a challenging task for embedded systems since it requires a lot of computational effort, while the power consumption must remain low. Therefore, the processor used must optimize the performance per Watt ratio, and this ratio is known to be better for small cores [1]. We believe that specific architectures like SIMD cores and GPU are suitable candidates for achieving the requested computation time while meeting the power constraints. This work considers several such architectures to perform movement detection and evaluate if they would fit the defined constraints.

This article aims to study and compare various implementations of two optical flow algorithms from a speed vs. energy consumption perspective, and highlights the impact of algorithmic and architectural optimizations for SIMD processors and GPUs.

Section II introduces the optical flow algorithms evaluated; section III presents all the optimizations implemented and algorithms variants; section IV presents the measurement protocol along with the target boards; section V presents and analyses the results obtained; and finally section VI concludes.

II. Optical flow iterative algorithms

More than a hundred optical flow algorithms exist, and vary according to their computational speed, quality of detection, and management of complex situations such as occlusions. An extensive qualitative analysis [2] of existing algorithms is available on the Middlebury's website [3], which also provides links to some of the codes. CMLA's IPol website [4] also provides various codes of recent algorithms. Optimized implementations of optical flow algorithms were the subject of numerous works on FPGA [5], [6], [7], [8] and on GPU [9], [10], [11], [12], but few on CPU [11], [13]. It should also be noted that optical flow estimations based on machine learning are gaining in popularity in the scientific community [14], [15].

This article focuses on two algorithms: the Horn-Schunck (HS) algorithm [16] and the TV-L1 algorithm [17]. While HS is not the most accurate, this well known algorithm has the advantage of simplicity. Its computation method is suitable for CPU and GPU implementations, and thus it can serve as a reference for more advanced algorithms which share its structure. Besides, the Horn-Schunck algorithm is used by the authors in a processing chain targeting the detection of meteors in real time from a nano-satellite [18]. TV-L1, which more computationally intensive, is studied because it can handle more complex situations such as night vision denoising.

The Horn-Schunck algorithm works as follows: it starts by computing the space-time derivatives (I_x, I_y I_t) of the pair of input images, then iteratively computes an average over a local neighborhood (eq.1), along with the update in every point of the velocity vector field (u, v) (eq. 3). The term α^2 is a convergence parameter.

$$\bar{u}^k = \frac{1}{12}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 0 & 2 \\ 1 & 2 & 1 \end{bmatrix} * u^k \qquad \bar{v}^k = \frac{1}{12}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 0 & 2 \\ 1 & 2 & 1 \end{bmatrix} * v^k \quad (1)$$

$$\begin{aligned} u^{k+1} &= \bar{u}^k - I_x \times \frac{I_x \times \bar{u}^k + I_y \times \bar{v}^k + I_t}{\alpha^2 + I_x^2 + I_y^2} \\ v^{k+1} &= \bar{v}^k - I_y \times \frac{I_x \times \bar{u}^k + I_y \times \bar{v}^k + I_t}{\alpha^2 + I_x^2 + I_y^2} \end{aligned} \quad (2)$$

TV-L1, which is well described in [17] and [4], relies on the total variation method using the L^1 norm for the regularization terms (TV-L1). It allows to have better estimation of discontinuities in the flow and the method is also more robust to the noise. This makes it particularly suitable for applications such as video denoising where the optical flow method needs to be robust to noise [19] and where a lot of occlusions can occur. TV-L1 estimation has however a greater computational demand compared to HS, as it uses different intermediate vector fields $\mathbf{v} = (v_1, v_2)$ and $\mathbf{P} = (P_{1x}, P_{1y}, P_{2x}, P_{2y})$ to compute the final field $\mathbf{u} = (u_1, u_2)$. Therefore, there are 3 major steps to compute one iteration of TV-L1:

$$\mathbf{v}^{k+1} = \mathbf{u}^k + TH(\mathbf{u}^k, \mathbf{u}^0) \quad (3)$$

$$\mathbf{u}^{k+1} = \mathbf{v}^{k+1} - \theta \mathrm{div}(\mathbf{P}) \quad (4)$$

$$\mathbf{P}^{k+1} = \frac{\mathbf{P}^k + \tau/\theta \nabla(\mathbf{v}^{k+1} + \theta \mathrm{div}(\mathbf{P}^k))}{1 + \tau/\theta \left|\nabla(\mathbf{v}^{k+1} + \theta \mathrm{div}(\mathbf{P}^k))\right|} \quad (5)$$

Where TH($\mathbf{u}$) is a conditional thresholding function. τ and θ are respectively the time step and tightness regularization terms.

Based on these equations, we can describe the algorithm with 3 major operators, each operator having a specific consumer/producer behavior, as presented in Figure 1.

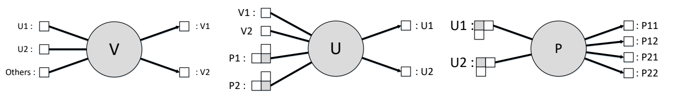

Fig. 1. Consumer/Producer representation of the main operators involved in TV-L1 optical flow estimation.

Using this representation highlights the difficulty to merge those operators since the computation of $u_{i,j}$ needs top and left points from $\mathbf{P}$ and the computation of $\mathbf{P}$ needs bottom and right points from $\mathbf{u}$. This is due to the computation of $div\mathbf{P}$ and $\nabla\mathbf{u}$. These considerations are important to understand the *pipeline* optimization presented in section III.

In order to simplify the performance analysis, the mono-resolution version of both algorithms are evaluated. Indeed, It gives good estimations in terms of computation time and power consumption as the hierarchical version empirically proves to be slightly less than twice slower. Besides, this allows to decouple the analysis from other concerns and choices, such as the interpolation method (bi-linear or bi-cubic), the number of scales, or the scaling method (Burt filter, Gaussian filter or a simple binomial filter). For the same reason, no warp [4] is considered for both algorithms. Large displacements require a multi-resolution version.

The qualitative aspects of these algorithms will not be studied in this article, as it puts the focus on the speed *vs.* power consumption trade-off for a real time detection with limited power. However, these algorithms have been chosen and are relevant to evaluate because the HS algorithm provides a reference baseline with good results for simple scenes, while the TV-L1 algorithm is able to handle more complex scenes with occlusions.

Lastly, a choice has to be made regarding the number of iterations: we decided to consider eight iterations per pixel for the HS algorithm, and three iterations for the TV-L1 algorithm, noticing experimentally that the corresponding Mean Squared Errors (MSE) were similar for these configurations. However, the results for both algorithms cannot be directly compared as their application domains are not identical.

III. Speed and Power Consumption Optimisations

Embedded system optimization for speed [20] and for power consumption [21], [22], [23] can be achieved by minimizing one of the two conditions under the hypothesis that the other follows a hard constraint. The proposed approach consists here in exploring the efficient frontier of the operating points in the (speed, consumption) space for codes with different levels of optimization, on different processors running for various frequencies. The points of compromise thus found can be used for example to define the maximum image size which can be processed by a configuration, and to help fuel the debate on power requirements.

The algorithmic transformations considered here for optimization are the ones described in [24], especially operators pipelining and operators fusion.

The HS and TV-L1 algorithms lend themselves well to this type of transformations: it is possible to merge the computation steps of the point-wise operators such as the V and U operators in Figure 1 for TV-L1. In order to reduce the number of memory accesses and to improve the arithmetic intensity, the result of V is no more stored in an array but directly used to compute the result of U. More importantly, the iterative part allows to pipeline iterations so that the accesses to a same memory cell are as close as possible in time. This way, data are more likely still in cache when they are accessed. Doing this requires to create intermediate arrays to store the results for each iteration.

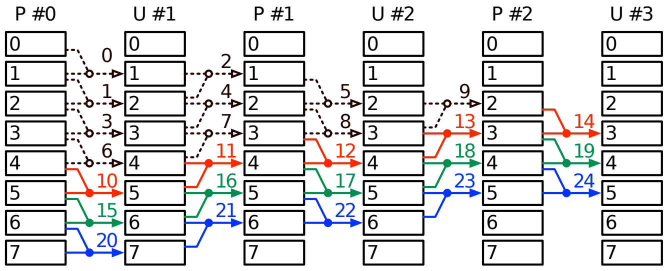

Fig. 2. Processing order for pipelined TV-L1 for 3 iterations (X #Y represents the data X at iteration Y). Dashed steps represent the prologue.

Figure 2 describes the pipeline optimization for 3 iterations of TV-L1 including a description of the prologue needed to deal with the images edges.

By analyzing the data dependencies within this transformation, we can first realize that all intermediate arrays can have modular memory allocation in order to reduce the number of lines per array to the number of iterations. With few adjustments of the prologue and the epilogue, we can even merge all the buffers so that we only have one buffer per vector field and the data modifications are made in-place.

These transformations maximize data persistence inside the processors (in the cache for CPUs, in the shared memory for GPUs) while minimizing transfers with the external memory. Similar transformations can be applied on the HS algorithm.

IV. Experimental evaluation

The five target boards used for experiments are presented in table I, three of which own a GPU. For each architecture, different implementations have been studied. For the CPU architectures, all the versions are multi-threaded with as many threads as physical cores. The considered variants for HS algorithm are: the base algorithm (*base*), two SIMD non-pipelined versions with respectively one or two buffers (*simd1* and *simd2*), and two pipelined SIMD versions with one or two buffers (*pipe 1* et *pipe 2*). For GPU architectures, two versions are considered: the baseline version (*base*) and an optimized version (*opt*) comprising the following optimizations: shared memory, grouping of U and V, operator merging and iterations kept within block. Additionally, these two versions are declined in two flavors: F_{32} et en F_{16}. As for the TV-L1 algorithm, nine versions have been tested, but for the sake of clarity we present here only four of them. The first one is the standard version with scalar implementation (*base*), the second one is its SIMD equivalent named *simd*, the last two ones are respectively called *pipe* and *pipe_mono* and both have SIMD, fusion and pipeline optimizations described in section III, the difference being that *pipe_mono* also has merged buffers (i.e. no intermediate buffer).

In order to perform simple and reproducible power measurements, the electrical consumption of the entire embedded system is measured. A board was developed to this effect which is inserted between the power source and the target board. Voltage, through a voltage divider, and current, through a resistor in series and a voltage amplifier, are measured by a micro-controller. The latter also reads the state of 4 GPIOs from the target board, in order to match measurement points and events from the computation program. The micro-controller sends 5000 samples per second to a PC host. The power measurement board was also carefully calibrated.

In the case of GPUs, performances are evaluated for each *kernel* [25] for blocks of 64 to 512 *threads*, using multiples of 32 (the size of a *warp* [25]). The blocks are from 8 to 256 *threads* large and 1 to 32 *threads* high. The best block size is then used within the comparisons.

TABLE I
Technical specifications of the target boards.

Board	Process	CPU	Fmax (GHz)	GPU	Fmax (MHz)
PCduino8	28 nm	8×A7	1.80	-	-
RPi3	40 nm	4×A53	1.20	-	-
TK1	28 nm	4×A15	2.32	192 C Kepler	852
TX1	20 nm	4×A57	1.73	256 C Maxwell	998
TX2	16 nm	4×A57	2.00	256 C Pascal	1300

We simultaneously measured performance and power consumption for different frequencies and image sizes. The frequencies are taken among the available frequencies on each board. Whenever possible, the external memory frequency is set to its maximum. The cooling system of each target board is also set to the maximum and the energy savings processes in the OS are deactivated. When the target board does not contain an integrated cooling down system, we performed the measures in an isothermal oven. We used all the ARM cores on each board, but we deactivated the Denver 2 cores on the Jetson TX2 board. The images used for the experiments are square images, with a varying size from 64 and 1024 pixels with a 8 pixel increment for the CPU versions; and with a varying size from 208 to 2048 pixels with a 16 pixel increment for the GPU versions. The size of 1008×1008, often used, corresponds to the largest size which is not a cache line size multiple and which is compatible with the GPU splitting.

In the case of GPUs, performances are evaluated for each *kernel* [25] for blocks of 64 to 512 *threads*, using multiples of 32 (the size of a *warp* [25]). The blocks are from 8 to 256 *threads* large and 1 to 32 *threads* high. The best block size is then used within the comparisons.

V. Results analysis

A. Operating points and efficient frontier

Figures 3 and 4 show the efficient frontiers of the operating points in the (time per pixel, energy consumption per pixel) space for each of the studied configurations. Figure 3 presents the results for the HS algorithm on CPUs and GPUs, while Figure 4 shows those for TV-L1 on CPUs.

The best configuration is obtained with the TX2 board for both HS (for both the GPU and the CPU versions) and TV-L1 algorithms. Figure 3 highlights the significant performance improvements brought by the GPU versions, and the impact of the presented optimizations: focusing on the TX2, compared to the base CPU version, the optimizations roughly bring a factor of 2.3 in speed, and the optimized GPU version (F_{32}) a factor 6 in speed and 9 in power consumption. We also note that for each configuration, the efficient frontier is generally small, showing that only a few of the possible frequencies are interesting w.r.t. the optimization criteria – typically between one and 3 points among 10 tested frequencies. This last aspect is even more significant for TV-L1 as we can see on Figure 4 that the efficient frontier is often reduced to a single point (usually the highest frequency). This figure also shows that the regular implementation of TV-L1 is far less efficient than the HS one. However, the optimizations made on TV-L1 bring a better improvement with a speedup of $\times 4.5$ and a consumption drop of more than $\times 6$. Still, even if the execution time of TV-L1 tends to reach the one of HS for the best boards (TX2 and TX1), its consumption remains significantly higher.

B. Impact of CPU optimizations

Figure 5(a and c) shows the impact of the considered CPU optimizations on the processing speed for the Jetson TK1 board – its behavior is well representative of all the other tested configurations on CPUs. By varying the image size, we can observe a cache exit generally sooner for the non-optimized

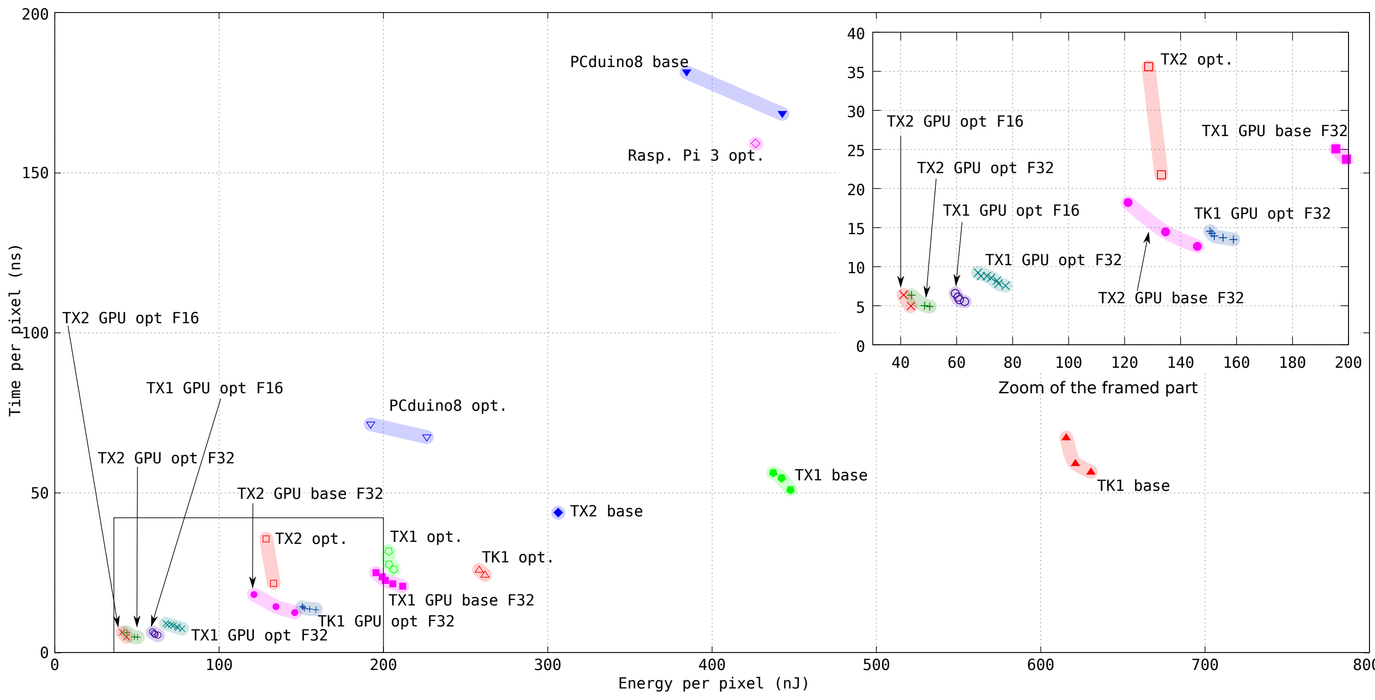

Fig. 3. Efficient frontier of the operating points in the (time/pixel, energy/pixel) space, for the considered target boards and HS implementations. Each point represents a given frequency. The *opt.* version for the general processors is the fastest version (*pipe 1*). The Raspberry Pi 3 *base* configuration is located outside of the represented space (Energy = 1145 nJ, Time = 415 ns). Bottom and left are better.

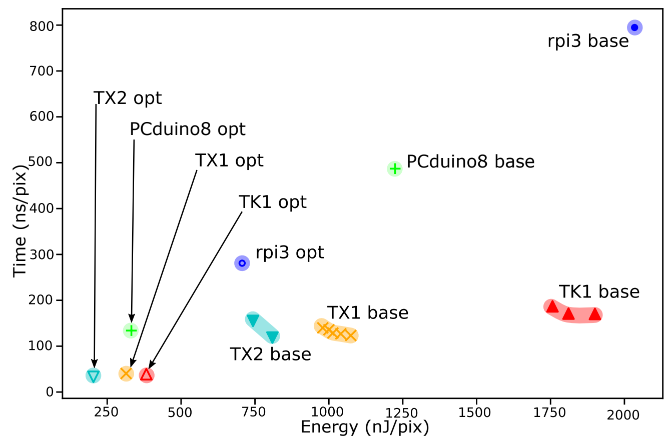

Fig. 4. Efficient frontier of the operating points in the (time/pixel, energy/pixel) space, for the considered target boards and TVL1 implementations. Each point represents a given frequency. Bottom and left are better.

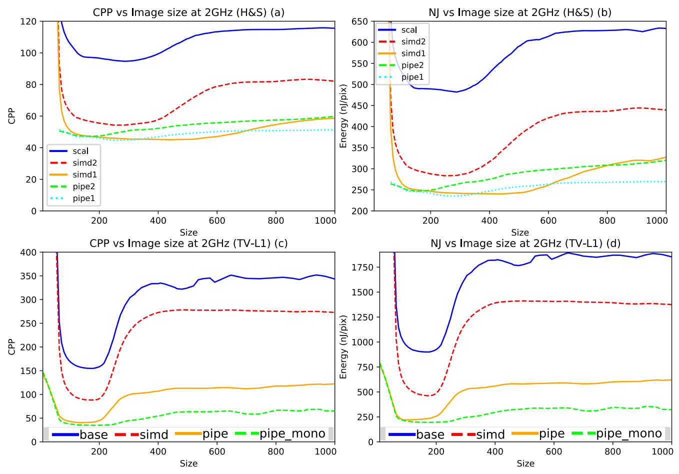

Fig. 5. Impact of the optimizations on CPU for both HS and TV-L1 on target board TK1. CPP = Cycles Per Pixel, Size = side length of a square image. Lower is better.

versions, particularly emphasized in the *simd1* version of HS (Figure 5(a)). Furthermore, among the best optimizations, the impact of the cache exit remains less important than on the other versions. As for example, on Figure 5(c), we can observe than for the *tvl1_simd* version, the cache exit is penalized by being 2.5 times slower than before when on the *tvl1_pipe* version, the penalty is only of a factor 2, while starting from a 2 times faster speed. We figured out that some optimizations slightly increase the power consumption – like the use of the SIMD units for HS – but Figure 5(b and d) shows that the time saved for processing one image with these optimizations still requires a lot less energy to process the same image. Besides, the most efficient optimizations (pipe versions for each algorithm) not only speed up the computation, but also reduce the power consumption, even with SIMD use. This little drop in power consumption could be explained by the fact that the pipe versions reduce the number of memory accesses and improve data persistence in cache, what saves both energy and time.

C. Impact of GPU Optimizations

Figure 6 shows the evolution of the number of cycles per pixel (CPP) and of energy as functions of frequency for the TX1 target board, on baseline and optimized codes, both in F_{16} and F_{32}. Figure 6(a) shows that in F_{32}, the *opt* version is about 3 times faster than the *base* version. The F_{16} *base* version is 1.5 to 2 times faster than the F_{32} *base* version. The opt F_{16} version is faster than the F_{32} only on high frequencies as the

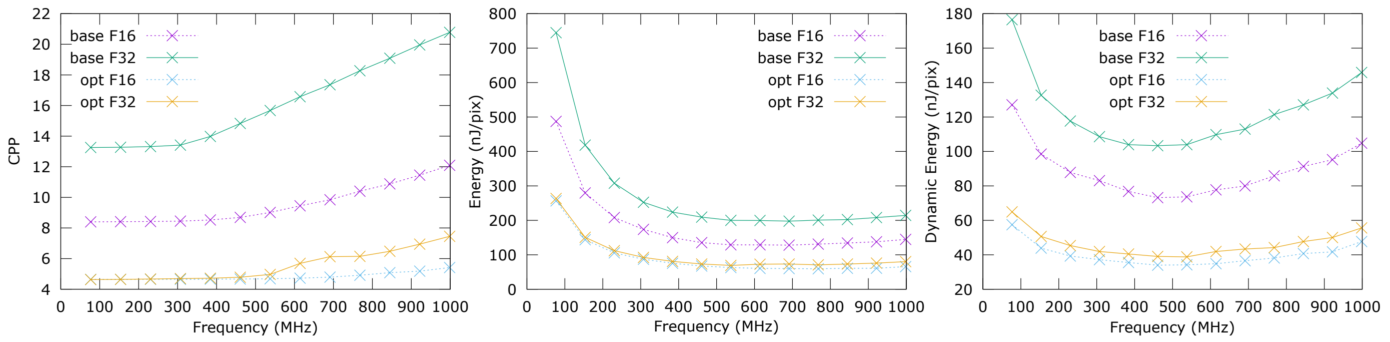

Fig. 6. CPP and power consumption as functions of frequency for the GPU TX1 versions. All image sizes are 1008×1008. Lower is better.

memory is no more fast enough compared to the processor. Because of the lack of ILP (Instruction Level Parallelism), F_{16} becomes interesting when the memory becomes a bottleneck. This is the same phenomenon that can explain the raise of CPP at high frequencies: the required bandwidth to maintain the same speed reaches the maximum target board bandwidth; the programs then becomes *memory bound*.

Figure 6(b) shows that the trend in power consumption is the same as for the speed: the variation in power consumption is negligible before the power consumption duration. On figure 6(c), the idle power consumption is subtracted from the total power consumption. We can notice that the frequencies minimizing the power consumption correspond to the *Compute Bound – Memory Bound* transition on which the CPP starts to grow. However, it is better to go towards higher frequencies if we consider the idle power consumption

D. Comparison of studied target boards and cores

Figure 7(a) shows that GPUs are faster than CPUs at a same frequency, achieving a ×4 factor on the TX1 board at 1GHz: the HS algorithm thus manages to take advantage of the greater number of cores. The GPU remains even faster at 1GHz than the CPU at 2GHz. The CPU of the TX1 exhibits a speedup compared to the one of the TK1, but remains close. The GPU from the TX1 improves significantly the performances compared to the GPU from the TK1 (factor 2). Its maximum frequency is also higher.

Figure 7(b) shows that the instantaneous power consumption of the different CPUs varies almost linearly w.r.t. the frequency, but with different slopes. The power consumption of GPUs is higher than on CPUs at an identical frequency, but GPUs can reach lower frequencies in order to reduce their power consumption. We also measured the spent energy per pixel for each board in order to make a fairer comparison (figure 7(c)). This allows us to choose an optimal operating frequency for each of the processor and to see that the evaluated GPUs are energetically more effective than the CPU.

E. Comparison between Horn-Schunck and TV-L1

Figure 8 presents a comparison between the two algorithms on a 1008×1008 image on the 3 targeted boards TK1, TX1 and TX2 depending on the frequencies. The comparison is made in terms of execution time (a), and energy consumption (b). It is made for the best version for each algorithm: *pipe1* for HS and *simd_piep_mono* for TV-L1. Even if the optimized version of TV-L1 takes advantage of a greater acceleration compared to its scalar version, it is still 1.4 time slower than HS (a). The power consumption of the two algorithms is quite the same, however we can notice that TV-L1 power consumption is always a little higher. This could be explained by the greater number of memory buffers used by TV-L1 which leads to more memory accesses.

F. Synthesis

The best configurations on CPU and GPU are obtained on the TX2 target board. Table II sums up these configurations and gives the largest image size possibly processed at 25 fps.

TABLE II
BEST CONFIGURATIONS OBTAINED, TO GET REAL-TIME AT 25 FPS.

Configuration	E (nJ/pix)	t (ns/pix)	max size (#pix)
HS TX2 GPU opt F_{16} energy min.	41	6.4	2501
HS TX2 GPU opt F_{16} time min.	43	5.0	2839
HS TX2 CPU opt energy min.	129	36	1059
HS TX2 CPU opt time min.	133	22	1355
TVL1 TX2 CPU opt global.	180	29	1174

VI. Conclusion

This article presents a comparison between different implementations of two iterative optical flow algorithms: Horn-Schunck and TV-L1. It demonstrates the superiority of GPUs over CPUs both in terms of computational speed and energy consumption for this kind of algorithm.

For HS, real-time at 25 f.p.s. is achieved for up to 8Mpix frames for 0.35J per picture on GPU, against 1.8Mpix frames for 0.24J per frame on CPU. The results also show that the TV-L1 algorithm, which is far slower than HS without optimization, can almost match the performance of HS after optimization on CPU, and can achieve real-time performances with 0.25J for a 1.4Mpix frame. These results can help an application developer to choose his algorithm and size his system. We also showed that even if HS remains faster than TV-L1, the latter can benefit more of major optimizations, what significantly reduces the differences between the two.

Future work will include the implementation of TV-L1 on GPU in order to have a more exhaustive comparison, and to evaluate the feasibility of implementing on CPU these algorithms in 16-bit fixed point in order to double the SIMD

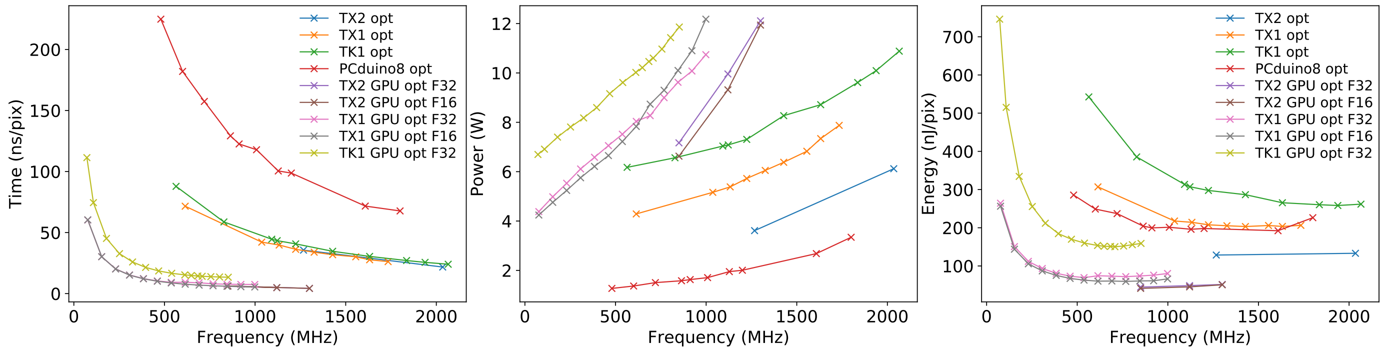

Fig. 7. Time per pixel (left), Power (middle) and Energy per pixel (right) as functions of Frequency on different target boards (optimized versions). The legend of the middle figure, omitted by lack of space, is identical. Lower is better.

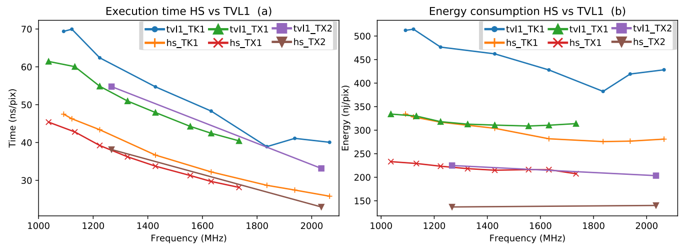

Fig. 8. Performances and consumption comparison between 8 optimized iterations of HS and 3 optimized iterations of TV-L1. Lower is better.

parallelism. A fine image qualitative analysis depending on the applications will also be made.

Acknowledgments

This work has been partially funded by the french DGA. The authors wish to thank Tomoko Arai from the PERC project in Chiba University for video sequences and JM Morel from CMLA at ENS Paris Saclay.

References

[1] S. Borkar, “Thousand core chips: a technology perspective,” in *Proceedings of the 44th annual Design Automation Conference*. ACM, 2007, pp. 746–749.

[2] S. Baker, D. S. J. P. Lewis, S. Roth, M. J. Black, and R. Szeliski, “A database and evaluation methodology for optical flow,” *International Journal of Computer Vision*, vol. 17,1, pp. 1–31, 2011.

[3] Middlebury, “Optical flow database http://vision.middlebury.edu/flow/.”

[4] IPOL, “Image processing on line http://www.ipol.im/.”

[5] G. Gultekin and A. Saranli, “An FPGA-based high performance optical flow hardware design for computer vision,” *Microprocessors and Microsystems*, vol. 37, no. 1-3, pp. 270–286, 2013.

[6] L. Bako, S. Hajdu, S.-T. Brassai, F. Morgan, and C. Enachescu, “Embedded implementation of a real-time motion estimation method in video sequences,” *Procedia Technology*, vol. 22, pp. 897–904, 2016.

[7] M. Kunz, A. Ostrowski, and P. Zipf, “An FPGA-optimized architecture of horn and schunck optical flow algorithm for real-time applications,” in *International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2014, pp. 1–4.

[8] Z. Chai, H. Zhou, Z. Wang, and D. Wu, “Using C to implement high-efficient computation of dense optical flow on FPGA-accelerated heterogeneous platforms,” in *International Conference on Field-Programmable Technology (FPT)*. IEEE, 2014, pp. 260–263.

[9] A. Plyer, G. L. Besnerais, and F. Champagnat, “Massively parallel lucas kanade optical flow for real-time video processing applications,” *Journal of Real-Time Image Processing*, vol. 11,4, pp. 713–730, 2016.

[10] J.D.Adarve and R. Mahony, “A filter formulation for computing real time optical flow,” in *(IEEE Robotic and Automation Letters)*. IEEE, 2016, pp. 1192–1199.

[11] T. Kroeger, R. Timofte, D. Dai, and L. V. Gool, “Fast optical flow using dense inverse search,” in *(ECCV)*, 2016.

[12] T. Senst, R. H. Evangelio, I. Keller, and T. Sikora, “Clustering motion for real-time optical flow based tracking,” in *Advanced Video and Signal-Based Surveillance (AVSS), 2012 IEEE Ninth International Conference on*. IEEE, 2012, pp. 410–415.

[13] A. Garcia-Dopico, J. L. Pedraza, M. Nieto, A. Pérez, S. Rodríguez, and J. Navas, “Parallelization of the optical flow computation in sequences from moving cameras,” *EURASIP Journal on Image and Video Processing*, vol. 2014, no. 1, p. 18, 2014.

[14] P. Weinzaepfel, J. Revaud, Z. Harchaoui, and C. Schmid, “Deepflow: Large displacement optical flow with deep matching,” in *International Conference on Computer Vision (ICCV)*. IEEE, 2013, pp. 1385–1392.

[15] A. Ranjan and M. J. Black, “Optical flow estimation using a spatial pyramid network,” in *(CVPR)*. IEEE, 2017, pp. 2720–2729.

[16] B. K. P. Horn and B. G. Schunk, “Determining optical flow,” *ACM Computing Surveys (CSUR)*, vol. 17, no. 1-3, pp. 185–203, 1981.

[17] C. Zach, T. Pock, and H. Bischof, “A duality based approach for realtime tv-l 1 optical flow,” in *Joint Pattern Recognition Symposium*. Springer, 2007, pp. 214–223.

[18] Météorix, “Nanosat Sorbonne Université http://www.nanosat.upmc.fr / fr/index.html.”

[19] C. Liu and W. T. Freeman, “A high-quality video denoising algorithm based on reliable motion estimation,” in *European Conference on Computer Vision*. Springer, 2010, pp. 706–719.

[20] K. Datta, M. Murphy, V. Volkov, S. Williams, J.Carter, L. Oliker, D. Patterson, J. Shalf, and K.Yelick, “Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures,” in *Supercomputing*. ACM/IEEE, 2008, pp. 1–12.

[21] R. Basmadjian and H. de Meer, “Evaluating and modeling power consumption of multi-core processors,” in *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*. ACM, 2012, p. 12.

[22] G. Hager, J. Treibig, J. Habich, and G. Wellein, “Exploring performance and power properties of modern multi-core chips via simple machine models,” *Concurrency and computation: practice and experience*, vol. 28, no. 2, pp. 189–210, 2016.

[23] S. Mittal and J. S. Vetter, “A survey of CPU-GPU heterogeneous computing techniques,” *ACM Computing Surveys*, vol. 47,4, pp. 1–36, 2015.

[24] L. Lacassagne, D. Etiemble, A. Hassan-Zahraee, A. Dominguez, and P. Vezolle, “High level transforms for SIMD and low-level computer vision algorithms,” in *ACM Workshop on Programming Models for SIMD/Vector Processing (PPoPP)*, 2014, pp. 49–56.

[25] NVIDIA. Cuda, programming model http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#programming-model.