Using HLS for Designing a Parametric Optical Flow Hierarchical Algorithm in FPGAs

Ilias Bournias, Roselyne Chotin, Lionel Lacassagne

Using HLS for Designing a Parametric Optical Flow Hierarchical Algorithm in FPGAs

Ilias Bournias*, Roselyne Chotin* and Lionel Lacassagne*
*Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
firstname.surname@lip6.fr

***Abstract*—In this work HLS is used for designing a parametric optical flow Hierarchical algorithm in FPGAs. The algorithm that is designed is the Hierarchical (pyramid) Horn and Schunck algorithm, both a multi-rate and multi-level (multi-scale) algorithm, which achieves larger motion displacement detection than the mono-scale ones. With the help of HLS, we parametrize our design in terms of the levels of the pyramid, the iteration factor and the number of pixels computed per clock. We are reusing the same resources in each level of the pyramid to keep the usage of DSPs and RAM low. We perform a design space exploration of the algorithm and we show that our fastest design achieves a throughput of 461 Mpixel/s in a 2048×2048 resolution pixel image.**

I. Introduction

Optical flow is used to compute the motion of the pixels of an image sequence. These kind of algorithms have a large number of applications ranging from object detection [1] to video denoising [2]. They are categorized according to the application they are used to, the accuracy that is needed and the throughput that has to be achieved. Hierarchical optical flow algorithms are gaining significant attendance during the last few years due to the limited motion estimation of the mono-scale algorithms (1 pixel). With hierarchical optical flow, larger pixel displacements can be detected, but that comes with a cost in the design as more hardware resources are needed [3].

For the implementation of hierarchical optical flow algorithms CPUs [2], GPUs [4]–[7] and FPGAs have been considered. Especially for FPGAs, Barranco et al. implemented the multi-scale (hierarchical) Lucas-Kanade ($L\&K$) algorithm [8], while Tomassi et al. remained on the multi-scale Phase-based algorithm [9] and Bournias et al. in the multi-scale Horn and Schunck optical flow algorithm [3].

However, all of the previous FPGA works, which face throughput and RAM bottlenecks, are not parametrized designs in terms of motion displacement (levels of the pyramid), accuracy (iteration factor), throughput (parallelization) and input image sizes, all of which are crucial for an optical flow algorithm design. Moreover, existing image processing compilers for FPGAs such as Halide-HLS [10] which does not support multi-rate image processing applications without the inter stage FIFO approach which highly increases the usage of block rams and Clockwork [11], which does not support efficiently multi-level (hierarchical) algorithms can not be used effectively for the designing of the algorithm. Thus, in this paper we are using HLS to design and parametrize a hierarchical optical flow algorithm in FPGAs, Horn and Schunck, which is both a multi-rate and multi-level (the images in different levels have different dimensions) algorithm and which to the best of our knowledge has never been done before.

A. Horn and Schunck algorithm

The basic scheme of Horn and Shunck ($H\&S$) [12] is an iterative algorithm (Fig. 1(a)) that estimates (u, v) from the first spatio-temporel derivatives I_x, I_y, I_t (of a pair of images) and from the previous average values $(\bar{u}, \bar{v})$, according to (1) and (2) where α is a smoothing parameter.

$$u^i = \bar{u}^{i-1} - I_x \frac{I_x \bar{u}^{i-1} + I_y \bar{v}^{i-1} + I_t}{\alpha^2 + I_x^2 + I_y^2} \tag{1}$$

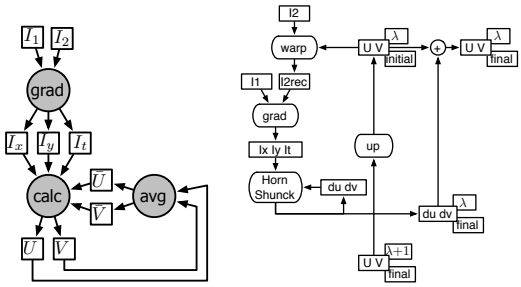
$$v^i = \bar{v}^{i-1} - I_y \frac{I_x \bar{u}^{i-1} + I_y \bar{v}^{i-1} + I_t}{\alpha^2 + I_x^2 + I_y^2} \tag{2}$$

As the derivatives are estimated with a $2 \times 2 \times 2$ kernel, the computed velocities should be smaller than 1 pixel / frame. That is the reason why, multi-scale (aka hierarchical) scheme should be considered (Fig. 1(b)).

From the computed velocities $(u, v)^{\lambda+1}_{final}$ of level $\lambda + 1$, a new velocity field is initialized : $(u, v)^{\lambda}_{init}$ by up-scaling the previous one with a factor 2, and multiplying it also by a factor 2: $(u, v)^{\lambda}_{init} = 2 \times \text{Upscale}(u, v)^{\lambda+1}_{final}$. These velocities are used to compensate (warp) the motion between the two images (I_{2rec}), thanks to a bi-linear or bi-cubic interpolation. Then $H\&S$ kernel iterates to provide the residual velocities $(\delta u, \delta v)$. After the iterations these residuals are accumulated to the initial estimation: $(u, v)^{\lambda}_{final} = (u, v)^{\lambda}_{init} + (u, v)^{\lambda}$ to provide the final velocity estimations at this level. Then same computations are done for the next level: $(u, v)^{\lambda-1}_{init} = 2 \times \text{Upscale}(u, v)^{\lambda}_{final}$ and so on until level $\lambda = 0$. The algorithm is explained thoroughly in the work of [3].

B. Contributions

In this paper, we are using HLS to design the hierarchical $H\&S$ algorithm in FPGAs, which is a multi-rate and a multi-level algorithm. We make our design parametric concerning the number of the levels of the pyramid, the iteration factor, the size of the image, the parallelization and two interpolation types (warping), in order to provide the optical flow designer with a lot of choices. Designs with a $\times 2$-factor for the iterations (20,10,5 and 40,20,10,5), for a 3 and a 4 level pyramid are implemented, which achieve a motion estimation in the range of



(a) H&S mono-scale iterative scheme (b) H&S multi-scale iterative scheme at level λ

Fig. 1. Description of the Algorithm

$(V,U) < |7|$ and $(V,U) < |15|$ respectively. We are tackling the problem of the warping [3] by re-using the same shift registers and DSPs in every pyramid level, a technique which we also apply to all our components to reduce the usage of block RAMs and DSPs in our design. Following that, we perform a design space exploration considering the number of the $H\&S$ cores used and we compare our work with other state of the art works proving that our biggest design outperforms in terms of throughput all the previous optical flow algorithms considering single precision IEEE floating point formats.

II. Proposed Architecture

A. Horn and Schunck core

Algorithm 1 shows the pseudo code of the $H\&S$. The core is parametric in terms of the levels of the pyramid (lvl), the size of the image (height, width), parallelization (par) [3] iteration factor (itf) and iteration number in level 0 (it_0). The delay line responsible for providing the core with the the 3x3 neighbored velocities (u_{00}^{i-1} to u_{22}^{i-1}) (dlin_U) is implemented with shift registers [13]–[15] and is reused in every pyramid level as it is also done with the DSPs. All the previous multi-scale implementations and the image processing compilers [10], [11] use dedicated registers and DSPs in every level which increases the usage of block RAMs ALUTs and DSPs.

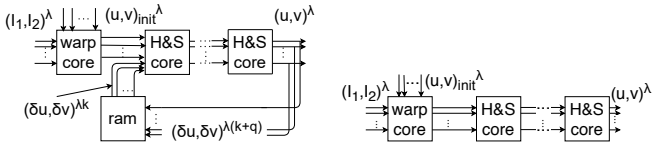
In this paper, for the pipelining and parallelization of all the $H\&S$ cores we propose two different designs to address the problem of throughput [3]. The partial pipeline parallel (P_{par}) and the fully pipeline parallel (F_{par}) mode as shown in Fig. 2 and as also used by [3] depending on the number of $H\&S$ cores of the design and the required number of iterations for each pyramid level. Par corresponds in the parallelization of the cores (pix/clk).

B. Warping core

Algorithm 2 shows the parametric pseudo code of the warping core with bi-cubic interpolation. As in the case of the $H\&S$ core the shift registers and DSPs are reused. For the bi-cubic (resp. bi-linear) interpolation, a neighbourhood of 4x4 (resp. 2x2) pixels in the input image is required. In order to ensure that one or more pixels are computed in every clock cycle (continuous streaming), all the required neighboured pixels for the interpolation have to be available [3]. This is

Algorithm 1 Pseudo code of the $H\&S$ core

```
define lvl, height, width, par, itf, it0;
```
Input: $u^{i-1}(par), v^{i-1}(par), I_1(par), I_{2rec}(par)$
Output: $u^i(par), v^i(par)$
```
hls_reg dlin_U[width·2+3+par]; //same for V
int l = lvl − 1;
```
while $l > 0$ **do**
  for `(i=0; h<`$2^{l\times 2^{itf-1}}\cdot it_0$`; i++)` **do**
    for `(h=0; h<`$\frac{height}{l+1}$`; h++)` **do**
      `#pragma unroll par`
      for `(w=0; w<`$\frac{width}{l+1}$`; w+=par)` **do**
        `#pragma unroll`
        for `(q=0; q<width·2+2; q+=par)` **do**
          dlin_U[q]=dlin_U[$q+par$] //same for V
        end
        dlin_U[$\frac{width}{l+1}\cdot 2+2+par$]=u$^{i-1}(par)$
        u$_{00}^{i-1}(par)$=dlin_U[par]; ...
        $u_{10}^{i-1}(par)$=dlin_U[$\frac{width}{l+1}+par$];
        ...$u_{22}^{i-1}(par)$=dlin_U[$\frac{width}{l+1}\cdot 2+par$];
        $u^i(par)$=H&S($u_{00}^{i-1}(par)$,
        ...,$I_1(par)$,$I_{2rec}(par)$); //same for V
      end
    end
  end
  $l--$;
end



(a) Partial Pipeline Parallel (P_{par}) Mode (b) Fully Pipeline Parallel (F_{par}) Mode

Fig. 2. Different Implementations Used

achieved with cub_m and cubf_m which guarantee that all the possible neighbourhoods of pixels are available. In order to choose the right neighbouring pixels in the two cases, the integer parts of the velocity vectors are needed (iu, iv) and the interpolation is done with the fraction part of the velocities (fu, fv).

C. Down-sampling, Sum, Up-scaling core

In the beginning of the algorithm each image used for each level of the pyramid is being built after the convolution from the coarser pyramid level with a 5x5 Gaussian kernel (down-sampling). The sum core is used to sum the velocities calculated in the previous levels with those of the current level. Then, results from the sum are extended (up-scaling) in order to be used in the next pyramid level. All the three cores are designed with HLS in the same way as the $H\&S$ with reusable shift

Algorithm 2 Pseudo code of the warping core (bi-cubic)

```
define lvl, height, width, par, itf, it0;
Input: I2(par), u(par), v(par)
Output: I2rec(par)
int l = lvl − 1;
hls_reg dlin_im[width · (2^(l+2) − 1) + 2^(l+2) + par];
float cub_m[2^(lvl+1)][4][par];
float cubf_m[4][4][par];
int iu[par]=(int)u[par]; //same for v
float fu[par]=u[par]−iu[par]; //same for v
while l > 0 do
    //same loops as in the H&S core
    dlin_im[width · (2^(l+1) − 1) + 2^(l+1) + par]=I2(par);
    #pragma unroll
    for (n=0; h<2^(lvl+1)/2^l; n++) do
        #pragma unroll
        for (m=0; h<4; m++) do
            cub_m[n][m][par]=
            dlin_im[n · width/2^l + m + iu + 2^lvl/2^l + par];
        end
    end
    #pragma unroll
    for (n=0; h<4; n++) do
        #pragma unroll
        for (m=0; h<4; m++) do
            cubf_m[n][m][par]=cub_m[n][m+iv+2^lvl/2^l+par]
        end
    end
    I2rec(par)=bi-cubic(cubf_m(par),fu(par),fv(par));
    //end of the loops as in the H&S core
    l − −;
end
```

registers and DSPs. In the case of the sum core, the intermediate velocities between the levels are stored in the external memory. In the work of [3] they are stored in the on chip memory which becomes a bottleneck when large images are considered [3].

D. Pipeline and Parallelism in each Pyramid Level

Up-scaling, warping, $H\&S$ and sum can be performed in a pipeline way for each pyramid level as all of them are executed in the same loops. With this way, interpolated pixels do not need to be written back to the external memory and then read again, but they can be directly processed by the $H\&S$ core in a pipeline way. The connection between the components is done with the streaming interfaces provided by the HLS compiler which eliminates the usage of inter stage FIFOs.

III. Design space exploration

A prediction model has been created for performing a design space exploration of the algorithm according to the designer's requirements (lvl, height, width, it_0, itf, interpolation type and computation time) to see if the design is implementable. The prediction is based on the number of the $H\&S$ cores used by the design. Depending on the number of iterations in each level of the pyramid, these cores are used in partial pipeline parallel (P_{par}) or fully pipeline parallel (F_{par}) mode as shown in Fig. 2.

TABLE I
CORE RESULTS FOR 3 AND 4 LEVEL PYRAMID FOR A FREQUENCY OF 250MHZ AND A 2048X2048 PIXELS IMAGE

Core	pix/clk	ALUT	FFs	RAMs	DSPs
bi-cubic 3-lvl	1	70700	49659	63	67
	2	160495	93835	51	134
	4	192970	127277	31	268
bi-linear 3-lvl	1	16849	9520	68	10
	2	79998	21428	46	20
	4	89433	27544	24	40
H&S 3-lvl	1	1677	12606	38	23
	2	2890	33044	45	46
	4	5331	31832	87	92
up-sample 3-lvl	-	1210	17581	2	8
down-sample 3-lvl	4	25417	17804	52	100
bi-cubic 4-lvl	1	263152	157745	130	67
	2	541014	301348	97	134
	4	604371	381150	59	268
bi-linear 4-lvl	1	102720	59772	135	10
	2	207121	125241	102	20
	4	268510	185041	71	40
H&S 4-lvl	1	4111	18439	71	23
	2	7855	47721	102	46
	4	12041	59104	194	92
up-sample 4-lvl	-	1720	21491	4	8
down-sample 4-lvl	4	29104	32041	71	100

Depending on the number of cores (Π) used (we count 1 core with a parallelization of par pix/clk as par cores and not 1) and the working frequency (f), the total time T for the multi-scale algorithm calculation of the image, without the down-sampling (which takes less than 15% of the total time), can be estimated by eq. (3).

$$T = \frac{Height \cdot Width \cdot (\sum_{n=0}^{lvl-1} \frac{it_0 \cdot 2^{l \cdot 2^{itf-1}}}{4^{lvl}})}{f \cdot \Pi} \quad (3)$$

Given the level of parallelism (par), the total number of DSPs (N) can be estimated by eq. (4).

$$N = \frac{\Pi}{par} \cdot N_{HS_{par}} + N_{warp_{par}} + N_{sum} + N_{up} + N_{do} \quad (4)$$

where $N_{HS_{par}}$, $N_{warp_{par}}$, N_{sum}, N_{up} and N_{do} are respectively the numbers of DSPs of $H\&S$, interpolation, sum, up-scaling and down-sampling cores.

The bandwidth rate of the external memory required is given by eq. (5) where the two images are read and u,v are both read and written back to the external memory. WL is the bitwidth of the words used.

$$EMBR = f \cdot par \cdot (2 + 2 \cdot 2) \cdot WL \quad (5)$$

The Ram usage M in blocks is estimated by eq. (6).

$$M = \frac{\Pi}{par} \cdot M_{HS_{par}} + M_{warp_{par}} + M_{sum} + M_{up} + M_{do} \quad (6)$$

TABLE II
DESIGN SPACE EXPLORATION RESULTS FOR A FREQUENCY OF 250MHZ AND A 2048X2048 PIXELS IMAGE

Implem.	3-level v_1	3-level v_2	3-level v_3	3-level v_4	3-level v_5	3-level v_6	4-level v_7	4-level v_8	4-level v_9	4-level v_{10}
iter.	20,10,5	20,10,5	20,10,5	20,10,5	20,10,5	20,10,5	40,20,10,5	40,20,10,5	40,20,10,5	40,20,10,5
HS cores(Π)	5	10	20	5	10	20	5	5	10	20
pix/clk(par)	1	2	4	1	2	4	1	1	2	4
interp.	bi-cubic	bi-cubic	bi-cubic	bi-linear	bi-linear	bi-linear	bi-cubic	bi-linear	bi-linear	bi-linear
Mode	P^4,P^2,F	P^4,P^2,F	P^4,P^2,F	P^4,P^2,F	P^4,P^2,F	P^4,P^2,F	P^8,P^4,P^2,F	P^8,P^4,P^2,F	P^8,P^4,P^2,F	P^8,P^4,P^2,F
ALUT	110k(26%)	208k(49%)	263k(62%)	65k(16%)	137k(32%)	163k(38%)	329k(77%)	173k(41%)	307k(72%)	392k(92%)
RAM(Bl.)	322(12%)	341(13%)	533(20%)	326(12%)	346(13%)	546(20%)	578(21%)	581(22%)	719(27%)	1151(43%)
DPSs	296(19%)	478(31%)	842(55%)	239(16%)	364(24%)	614(40%)	296(19%)	239(16%)	364(24%)	614(40%)
EMB(GB/s)	9.3(27%)	15.2(44%)	27.4(80%)	9.1(26%)	14.9(43%)	26.8(78%)	9.3(27%)	9.1(26%)	15.4(45%)	27.1(79%)
fps	31	59	105	31	60	110	25	26	45	72
thr.(Mpixel/s)	130	247	440	130	251	461	104	109	188	301

TABLE III
COMPARISON TO STATE-OF-THE-ART IMPLEMENTATION ON FPGA (SORTED ON THE THROUGHPUT)

Implementation	Algorithm	size	format	Π	frame rate	Throughput (Mpixel/s)	Architecture
This work v_{10}	Multi-scale H&S 4-level	2048×2048	F_{32}	20	72	301	Arria 10 250Mhz
This work v_6	Multi-scale H&S 3-level	2048×2048	F_{32}	20	110	461	Arria 10 250Mhz
[9]	Multi-scale Phase-based	640×480	$Q_{8.0\rightarrow 8.4}$	-	31.5	9.6	Virtex-4 45MHz
[8]	Multi-scale L&K	640×480	$Q_{9.0\rightarrow 29.8}$	-	32.0	9.8	Virtex-4 83 MHz
[16]	Phase-based	512×512	$Q_{8\rightarrow 12}$	-	40.0	10.4	Virtex-4 42MHz
[17]	HBM + Refinement	640×480	-	-	39.0	12.0	Virtex-7 200MHz
[18]	Mono-scale H&S	1920×1080	F_{32}	32	96.5	200	Virtex-7 200MHz
[3]	Multi-scale H&S 3-level	1024×1024	F_{32}	10	202	211	Stratix V 208MHz
[19]	Mono-scale H&S	4096×2304	Q	20	30.0	283.1	Stratix IV -
[3]	Multi-scale H&S 3-level	1024×1024	F_{16}	20	484	507	Stratix V 278MHz
[20]	Mono-scale L&K	1024×1024	$Q_{10\rightarrow 32}$	-	1000.0	1048	Virtex-2 90 MHz

where M_{HS}, M_{warp}, M_{sum}, M_{up} and M_{do} are respectively the blocks used to store values of $H\&S$, warp, sum, up-scaling and down-sampling cores.

Given the desired requirements by the designer and the FPGA device's constraints, by solving eq. (3), (4), (5) and (6) our prediction model estimates the number of $H\&S$ cores, par, DSPs, block RAMs and external memory bandwidth rate usage. If the design is implementable in the specific device then the designer's requirements are inserted in the HLS tool to provide the final implementation.

IV. RESULTS OF IMPLEMENTATION

For the implementation of the algorithm, the FPGA Intel Arria 10 GX 1150 was used and the intel HLS compiler (version 20.2). The Arria 10 is equipped with 1518 DSPs and a peak external memory bandwidth rate of 34.4 GB/sec. The computation is done in single precision floating point IEEE format.

In Table I we can see the information about all the key components used in our design and which are used for our prediction model. It can be seen that the interpolation components request a lot of ALUTs because of the many multiplexers that have to be implemented to ensure the continuous streaming. The bi-cubic interpolation, which should be chosen when accuracy is important, with a par of 2 pix/clk and 4 pix/clk (maximum vectorization [13] we consider in this paper) for a 4 level pyramid is not implementable in the Arria 10 FPGA due to the high demand for ALUTs. The same happens with interpolations with a par of 8 pix/clk (we do not include them in table I). In Table II the results of the designs (we chose very standard configurations [2], [3], [21]) we implemented are presented. Our fastest 3-level and 4-level designs achieves a throughput of 461 Mpixel/s and 301 Mpixel/s respectively.

In Table III we make a comparison of our works with the state of the art Optical flow algorithms implemented in FPGA. We can see from this table that our designs outperform in terms of throughput all the other designs even the mono-scale ones, except of those of Bournias [3] and Ishii [20]. Bournias uses a F_{16} format (IEEE half precision) with limited accuracy. However, his F_{32} design is slower than ours, all of his designs does not support different parameters like ours and they also face a bottleneck of on chip RAM when images larger than 1024x1024 pixels are considered. Ishii is implementing a Mono-scale $L\&K$ algorithm with pseudo-variable frame rate by using two FPGAs and a PC to make the computation of the algorithm.

V. CONCLUSION

We used HLS to design a parametric hierarchical optical flow algorithm (both a multi-rate and multi-level algorithm), $H\&S$, which to the best of our knowledge has never been done before. Our designs are parametric in terms of the levels of the pyramid, the sizes of the image, parallelization, iteration factor and two interpolation types. Our biggest designs achieve comparable and better throughput compared to the other state of the art designs and at the same time they detect wider motion displacements. In the future we plan to explore different arithmetic formats in order to include them in our designs.

REFERENCES

[1] B. Johnson, S. Thomas, and R. Sheeba, "A high-performance dense optical flow architecture based on red-black sor solver," *Journal of Signal Processing Systems*, pp. 357–373, 2020.

[2] 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 *2019 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, 2019, pp. 47–52.

[3] I. Bournias, R. Chotin, and L. Lacassagne, "Fpga acceleration of the horn and schunck hierarchical algorithm," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2021, pp. 1–5.

[4] A. Garcia-Dopico, J. Pedraza, M. Nieto, A. Perez, S. Rodrigeuz, and J. Navas, "Parallelization of the optical flow computation in sequences from moving cameras," *EURASIP Journal on Image and Video Processing*, p. 18, 2014.

[5] A. Plyer, G. Le Besnerais, and F. Champagnat, "Massively parallel Lucas Kanade optical flow for real-time video processing applications," *Journal of Real-Time Image Processing*, vol. 11, pp. 713–730, 2016.

[6] T. Romera, A. Petreto, F. Lemaitre, M. Bouyer, Q. Meunier, and L. Lacassagne, "Implementations impact on iterative image processing for embedded gpu," in *2021 29th European Signal Processing Conference (EUSIPCO)*, 2021, pp. 736–740.

[7] M. Millet, N. Rambaux, A. Petreto, F. Lemaitre, and L. Lacassagne, "Meteorix - a new processing chain for real-time detection and tracking of meteors from space," *WGN, Journal of the International Meteor Organization*, vol. 49, no. 6, Dec. 2021.

[8] F. Barranco, M. Tomasi, J. Diaz, M. Vanegas, and E. Ros, "Parallel Architecture for Hierarchical Optical Flow Estimation Based on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 6, pp. 1058–1067, 2012.

[9] M. Tomasi, M. Vanegas, F. Barranco, J. Diaz, and E. Ros, "High-Performance Optical-Flow Architecture Based on a Multi-Scale, Multi-Orientation Phase-Based Model," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 12, pp. 1797–1807, 2010.

[10] J. Pu, S. Bell, X. Yang, J. Setter, S. Richardson, J. Ragan-Kelley, and M. Horowitz, "Programming heterogeneous systems from an image processing dsl," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 3, Aug. 2017. [Online]. Available: https://doi.org/10.1145/3107953

[11] D. Huff, S. Dai, and P. Hanrahan, "Clockwork: Resource-efficient static scheduling for multi-rate image processing applications on fpgas," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2021, pp. 186–194.

[12] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, no. 1, pp. 185 – 203, 1981.

[13] H. R. Zohouri, A. Podobas, and S. Matsuoka, "Combined spatial and temporal blocking for high-performance stencil computation on fpgas using opencl," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2018, p. 153–162.

[14] H. R. Zohouri, A. Podobas, and S. Matsuoka, "High-performance high-order stencil computation on fpgas using opencl," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2018, pp. 123–130.

[15] H. M. Waidyasooriya, Y. Takei, S. Tatsumi, and M. Hariyama, "Opencl-based fpga-platform for stencil computation and its optimization methodology," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 5, pp. 1390–1402, 2017.

[16] M. Tomasi, M. Vanegas, F. Barranco, J. Daz, and E. Ros, "Massive Parallel-Hardware Architecture for Multiscale Stereo, Optical Flow and Image-Structure Computation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 2, pp. 282–294, 2012.

[17] K. Seyid, A. Richaud, R. Capoccia, and Y. Leblebici, "FPGA-Based Hardware Implementation of Real-Time Optical Flow Calculation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 1, pp. 206–216, 2018.

[18] M. Komorkiewicz, T. Kryjak, and M. Gorgon, "Efficient hardware implementation of the Horn-Schunck algorithm for high-resolution real-time dense optical flow sensor," *Sensors*, vol. 14, no. 2, pp. 2860–2891, 2014.

[19] M. Kunz, A. Ostrowski, and P. Zipf, "An FPGA-optimized architecture of horn and schunck optical flow algorithm for real-time applications," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, 2014, pp. 1–4.

[20] I. Ishii, T. Taniguchi, K. Yamamoto, and T. Takaki, "High-Frame-Rate Optical Flow System," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 1, pp. 105–112, 2012.

[21] 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 *2018 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, 2018, pp. 25–30.