2011 21st International Conference on Field Programmable Logic and Applications

SEPARABLE FIR FILTERING IN FPGA AND GPU IMPLEMENTATIONS: ENERGY, PERFORMANCE, AND ACCURACY CONSIDERATIONS

Daniel Llamocca†*, Cesar Carranza***, *and Marios Pattichis*†

†Electrical and Computer Engineering Department,
The University of New Mexico, Albuquerque, NM, 87131, USA
*Sección Electricidad y Electrónica, Pontificia Universidad Católica del Perú, Lima 32, Perú
dllamocca@ieee.org, acarran@pucp.edu.pe, pattichis@ece.unm.edu

ABSTRACT

Digital video processing requires significant hardware resources to achieve acceptable performance. Digital video processing based on dynamic partial reconfiguration (DPR) allows the designers to control resources based on energy, performance, and accuracy considerations.

In this paper, we present a dynamically reconfigurable implementation of a 2D FIR filter where the number of coefficients and coefficients values can be varied to control energy, performance, and precision requirements. We also present a high-performance GPU implementation to help understand the trade-offs between these two technologies.

Results using a standard example of 2D Difference of Gaussians (DOG) filter indicate that the DPR implementation can deliver real-time performance with energy per frame consumption that is an order of magnitude less than the GPU. On the other hand, at significantly higher energy consumption levels, the GPU implementation can deliver very high performance.

1. INTRODUCTION

Hardware implementations of digital video processing methods are of great interest because of the ubiquitous applications. In terms of performance acceleration, many image and video processing algorithms require efficient implementation of 2D FIR filters [1].

Dynamic partial reconfiguration (DPR) allows FPGA designers to explore different implementations based on energy, performance, and accuracy requirements. In addition to DPR, efficient filter implementations are based on the direct use of LUTs [2], the distributed arithmetic technique [3], and separable designs [4, 5].

On the other hand, Graphic Processing Units (GPUs) offer high-performance floating point capabilities at significant energy consumption levels [6]. With the introduction of OpenCL and CUDA (Compute Unified Device Architecture), there has been a significant growth of GPU implementations; with [7] and [8] as examples of 2D FIR implementations.

.

In this paper, we are interested in exploring the energy, performance, and accuracy trade-offs between DPR FPGA and the corresponding GPU implementations. Some trade-offs have been explored in [8] and [9]. Our goal is to provide recommendations for different implementations based on specific energy and performance requirements.

The paper is organized as follows: Section 2 describes the embedded filter implementation on the FPGA. Section 3 details the GPU filter implementation. Section 4 explains the measurement setup for both implementations. Section 5 presents the results in terms of Energy, performance, and accuracy. Finally, Section 6 summarizes the paper.

2. 2D FIR FILTER SYSTEM ON THE FPGA

We consider a 2D separable filtering implementation that is an extension of prior work presented in [3] and [4]. Here, we extend prior research to allow DPR of the entire FIR core, its FSL (Fast Simplex Link) bus interface, and the Partial Reconfiguration Region (PRR) control interface.

2.1. System Architecture

Figure 1 depicts the block diagram of the embedded system. The 1D FIR Filter processor core and the PowerPC (PPC) interact via the Fast Simplex Link (FSL) bus. The PRR is reconfigured via the internal configuration access port (ICAP). The Compact Flash (CF) card holds the partial bitstreams and input data. Bus macros are no longer needed in the Xilinx ISE 12.2 Partial Reconfiguration Tools.

In the context of the embedded system of Fig. 1 a 2D separable filter is realized by i) filtering the rows, ii) turning a row filter into a column filter via DPR, and iii) filtering the columns. Figure 2 depicts this scheme, where the 2D filter can be modified at run-time by using a different pair of row and column filters.

The filtered images (row and column-wise) are stored in the DDRRAM. The row-wise filtered image is transposed in the memory before it is streamed to the column filter. The system is implemented in the ML405 Xilinx Dev. Board that houses a XC4VFX20 Virtex-4 FPGA. The PPC is clocked at 300 MHz and the peripherals at 100 MHz.

978-0-7695-4529-5/11 $26.00 © 2011 IEEE
DOI 10.1109/FPL.2011.71

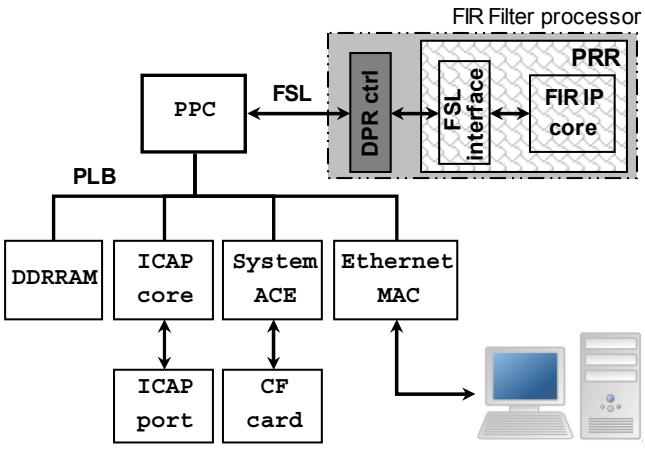

Figure 1. System Block Diagram

2.2. 1D FIR filter core

This fixed-point core is based on the one presented in [3]. We allow for full-reconfiguration, i.e. the entire filter is included in the PRR. Several modifications are introduced:

- A new parameter 'B' allows the specification of the input data bit-width. This is different from the coefficients' bit-width.
- The frame size determines the input length of both the row and column filter. The input length is a parameter to the Finite State Machine (FSM) that controls the FSL interface. As a result, the FSL interface has to be included in the PRR (see Fig. 1).
- An interface that disables the PRR outputs during reconfiguration is required since the PRR outputs now include FSL interface signals (shown in Fig. 1).

Here, the 2D filter requires 2 bitstreams: one for the row filter and one for the column filter. The PRR must accommodate the largest filter.

3. FILTER IMPLEMENTATION ON THE GPU

We consider a parallel FIR algorithm implementation in the CUDA environment [10]. Here, parallelism is achieved by a grid that consists of blocks, with each block having a number of threads. All threads within a block are run in parallel from the software perspective. The actual number of blocks that can run in parallel is bounded by the number of streaming multiprocessors (SMs). Here, we can run a single block on each SM. Also, the number of threads that can be run in parallel at each SM is given by the number of CUDA cores inside each SM.

For the purposes of this paper, we will report energy and performance measurements on the GPU (termed the device) as opposed to the CPU (termed the host). Here, GPU memory is divided into global memory, shared memory, constant memory, and texture memory.

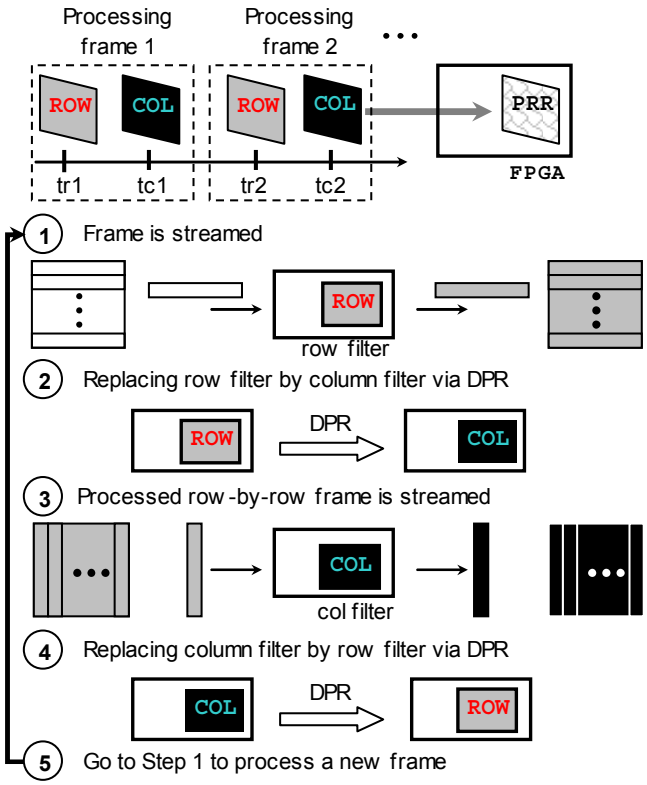

Figure 2. 2D separable FIR filter implementation.

The algorithm exposes and exploits parallelism of the 2D FIR filter in order to obtain significant speed up gains. It is based on ideas exposed in [7]. Double precision (64 bits) is utilized. The filter symmetry and its separability are taken advantage of. We summarize the algorithm steps below:

1. The image and the filter kernel are transferred form the host to the device (global memory).
2. The image is then divided into blocks. Each image block is filtered by a thread block by rows.
3. The row-filtered image is also divided into blocks. Each image block is column- filtered by a thread block.
4. The final filtered image is transferred to the host.

To further describe the algorithm, we let the input image to be of size HxW (H rows by W columns) and a filter kernel of size KxK (row and column filter of same length). We refer to [7] for more details on the separable implementation. Performance is achieved based on: i) loop unrolling, ii) storing image blocks in shared memory, and iii) storing the filtering coefficients in constant memory.

Each image block is processed as follows: It is first loaded to the shared memory (with extra $\lfloor K/2 \rfloor$ pixels on both sides for correct filtering). Then, for row filtering, each thread inside a block performs a point-wise multiplication between the row kernel and a row portion of the image; and then adds up each product producing an output pixel. This process continues until the filtered image block is obtained.

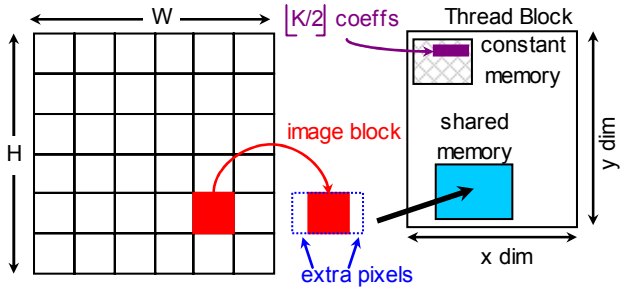

Figure 3. Thread block configuration for row filtering

Figure 3 shows the setup of a thread block for row filtering. Since all thread blocks work concurrently (from the software perspective), we are left with the row-filtering image in the global memory at the end of the previous process. This image (in blocks) is loaded again in shared memory, this time to perform column filtering. A thread block does not transpose the column-ordered data since the image block is small and it is not worth the effort. Thus, this division of the image in blocks effectively avoids transposing the entire image prior to column filtering.

For row processing, the dimension of the thread block in the x direction must be higher or equal than $\lfloor K/2 \rfloor$ (effective size of the kernel). For the dimension in y direction, any power of 2 is suitable as long as H is its multiple. For column processing, the dimension of the thread block in the y direction must be higher or equal than $\lfloor K/2 \rfloor$. For the dimension in x direction, any power of 2 is suitable as long as W is its multiple.

The device utilized is a NVIDIA GeForce GTX465, with 11x32 CUDA cores running at 607 MHz. There are 11 Streaming Multiprocessors that run at 1.215 GHz, each with 32 CUDA cores. 1 GB of GDDR5 memory is available and runs at 1603 MHz with a bandwidth of 102.6 GB/s. There are 48K bytes of shared memory per block. The maximum power dissipation of the board is 200 W.

The GPUs are tested in a desktop environment with an Intel® Xeon W3520 running at 2.67 GHz, with 6GB of DRAM. Our software configuration uses Windows 7 Ultimate (64-bits) with CUDA 3.2.

4. EXPERIMENTAL SETUP

This section details how the results were obtained. The set of filters for our test are first described. Then we detail how performance, energy, and accuracy were measured.

4.1. Set of filters for testing

To demonstrate the system, we consider a popular bandpass filter implementation based on the Difference of Gaussians (DOG) filter with $\sigma_1 = 2, \sigma_2 = 4$ [1]. For comparison, we implement the DOG filter using 48 coefficients and double precision arithmetic precision.

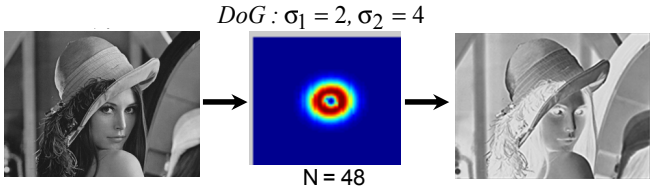

Figure 4. Frequency response – ideal filter with N = 48

The input image selected is the standard grayscale level (8 bits) image 'Lena'. Fig. 4 shows the ideal frequency response of the filter with the input and output images.

We consider 6 filter implementations, each with a different number of coefficients (N = 8, 12, 16, 20, 24, 32). In addition, we consider 3 different frame sizes: 640x480 (VGA), 352x288 (CIF), 176x144 (QCIF), derived from cropped versions of 'Lena' (to preserve the frequencies). This results in 18 filtered images.

In the case of the FPGA implementation, the bit-width of the coefficients is set at 16 bits. The row filter receives 8-bit pixels at the input and outputs 16-bit pixels. The column filter receives and outputs 16-bits pixels, taking advantage of the symmetry of the filter [3].

The system switches to a different 2D filter via DPR. This is realized by reconfiguring a different row filter at step 4 in Figure 2. Then, having streamed the image through the new row filter, we load the respective column filter at step 2. After that, we keep switching between these new row and column filters. As a result, the PRR size is that of largest column filter.

In the case of the GPU implementation, the system is implemented with double floating point numerical precision, although it can be programmed with fixed-point.

4.2. Energy, performance, and accuracy measurements

We measure performance in terms of frames per second (fps). In the case of the FPGA implementation, the processing time per frame includes: i) row filtering process, ii) column filtering process, iii) transposing row-filtered image, and iv) PRR reconfiguration (twice). The transposing of the row-filtered image occurs right after the filtering of the rows is completed. Two reconfigurations are needed per frame. Then, the performance (fps) is given by:

$$fps_{FPGA} = 1/\left(t_{rows} + t_{cols} + t_{transpose} + 2 \times t_{reconfig}\right) \quad (1)$$

In the case of the GPU implementation, the processing time per frame includes: i) Allocation of memory and data transfer from host to device, ii) Frame processing, and iii) data transfer from device to host. We run the filters 1000 times and get an average quantity of each of these times.

$$fps_{GPU} = 1/\left(t_{alloc+transf(h->d)} + t_{process} + t_{transf(d->h)}\right) \quad (2)$$

With regard to energy measurements, we consider the energy consumption per frame.

In the FPGA case, the power spent by the three Virtex-4 FPGA power sources (VCCINT, VCCAUX, VCCO) is obtained, which amounts to the embedded system power consumption. We use the Xilinx Power Analyzer (XPA) tool that provides a more accurate estimate than the Xilinx Power Estimator (XPE) because it is based on simulated switching activity of the place-and-routed circuit [11]. Our results are obtained with XPA at 25ºC. We get the power drawn by both the row (P_{row}) and column filter (P_{col}).

Each filter variation amounts to a difference in resource usage, and in turn in different power consumption. However, the filter core is small compared to the rest of the embedded system, so the power difference is not noticeable. As a result, it is more useful to consider the power drawn (both P_{row} and P_{cols}) just by the FIR Filter IP core.

The power consumption during reconfiguration is an important quantity since the 2D FIR filter makes intensive use of DPR. Unfortunately, there is no tool available that can provide an estimate of this power consumption. In [12], hardware measurements determined that only the VCCAUX supply current increased during reconfiguration, and it increased by 25 mA for the XC4VFX12 device. This dynamic current does not depend on the device size, so we use this current for our XC4VFX20 device. The reconfiguration power then results:

$$\begin{aligned} P_{reconfig-row} &= P_{row} + (25mA) \times VCCAUX \\ P_{reconfig-col} &= P_{col} + (25mA) \times VCCAUX \end{aligned} \quad (3)$$

Note that $P_{reconfig-row}$ is the power during reconfiguration of the row filter into a column filter. $P_{reconfig-col}$ is defined in a analogous fashion.

With the processing times of the row and column filter, and the reconfiguration time, the energy per frame results:

$$\begin{aligned} epf_{FPGA} = {} & P_{row} \times t_{rows} + P_{col} \times t_{cols} + \\ & \left(P_{reconfig-row} + P_{reconfig-col}\right) \times t_{reconfig} \end{aligned} \quad (4)$$

In the case of the GPU implementation, similarly to [6], the current is measured with the clamp sensor ESI 687 on the power connectors. Both the external power of the GPU and the power provided to the PCIe bus (20 W max.) are considered. Note that we measure the power consumption of the whole board that includes the GPU, memory, and other components. The average power during the tasks is measured, thus the energy per frame results:

$$\begin{aligned} epf_{GPU} = {} & P_{average-clamp} \times \\ & \left(t_{alloc+transf(h->d)} + t_{process} + t_{transf(d->h)}\right) \end{aligned} \quad (5)$$

Since the transferring and allocation times can be considered as an offset any GPU implementation has to deal with, we might also be interested in measuring the energy per frame spent only during the processing stage:

$$epf_{GPU} = P_{average-clamp} \times t_{process} \quad (6)$$

Table 1. Embedded FIR Filtering system resource utilization (Virtex-4 XCVFX20-11FF672)

Module	Slice	(%)	FF	(%)	LUT	%
PRR (col filter)	2125	25%	3680	21%	3812	22%
Static Region	4973	58%	5226	31%	5998	35%
Overall	7098	83%	8906	52%	9810	57%

For accuracy measurements, we define accuracy as the relative error between the FPGA or GPU processed frame and the results using double precision with 48 coefficients. Consequently, we measure accuracy using the PSNR between the FPGA or GPU outputs and the double precision implementation (48 coefficients). Here, note that GPU implementation is also using double precision but with variable number of coefficients. On the other hand, for the FPGA, the error is due to truncation in the number of coefficients and the use of fixed-point arithmetic (16 bits).

5. RESULTS

5.1. FPGA resource usage and reconfiguration time

The PRR must accommodate the largest filter (column filter with N = 32). Thus, the PRR occupies a tightly packed area of $24 \times 90 = 2160$ Virtex-4 slices with a bitstream size of 183754 bytes. It takes about 25% of the FPGA fabric.

Table 1 shows the hardware resource usage of the embedded FIR filtering system of Figure 1. It reveals the actual resource usage of the PRR and the static region. Note that the largest column filter (N = 32) occupies 2125 Slices (98% of the PRR Slices).

A reconfiguration speed of 3.28 MB/s is obtained with the Xilinx® ICAP core, resulting in 56.02 ms of reconfiguration time for the given bitstream size.

5.2. Running times

In the FPGA case, t_{rows} and t_{cols} are in line with the FSL transfer speed of 226 Mbps reported in [2]. For example, for N = 32, t_{rows} = 10971, 3620, and 905 us for the VGA, CIF, and QCIF frame sizes respectively. The number of coefficients plays a negligible role in the processing time because the FIR filter is a fully pipelined system in which the number of coefficients only increments the register levels, which in turn increases the initial latency of the pipeline (that fades out for an input length larger than the number of coefficients). This effect is usually masked by the bus speed with bus cycles larger than the register levels of the pipeline. System performance is limited by the time spent in transposing the image (about 4152 us, 1453 us, and 379 us for the VGA, CIF, and QCIF frame sizes respectively) and the reconfiguration time (about 56.02 ms).

The reconfiguration time of 56.02 ms achieved with the Xilinx® ICAP controller significantly limits real-time system performance. With the use of the custom-made

Table 2. GPU running times (ms). N: number of coefficients

	N	$t_{process}$	$t_{alloc+transf(h->d)}$	$t_{transf(d->h)}$
640x480	8	0.4099	2.0	1.9
	12	0.4661	1.9	1.8
	16	0.5096	2.0	1.6
	20	0.5801	1.9	1.8
	24	0.6481	1.9	1.9
	32	0.7777	1.9	1.8
352x288	8	0.2536	1.14	0.86
	12	0.3031	1.12	0.94
	16	0.3486	1.10	0.90
	20	0.3527	1.40	0.75
	24	0.3975	1.73	0.70
	32	0.4610	1.40	0.60
176x144	8	0.1998	0.60	0.30
	12	0.2105	0.75	0.35
	16	0.2371	0.60	0.30
	20	0.2417	0.70	0.30
	24	0.2729	0.80	0.30
	32	0.2853	0.80	0.30

ICAP controller presented in [13], the reconfiguration time would be 0.622 ms. For a good comparison with the GPU, this reconfiguration time is used instead. Note that the custom ICAP core has different power requirements than the Xilinx® ICAP core. In practice, we expect the power difference to be negligible since the custom ICAP core is a small (and low-power) circuit.

In the case of the GPU, we found that most of the time is consumed by the allocation of memory and data transfers from/to host to/from device. Table 2 shows these times.

Note that $t_{alloc+transf(h->d)}$ and $t_{transf(d->h)}$ are about the same for a given frame size. Also, the processing times do vary according to the number of coefficients, unlike in the case of the FPGAs.

5.3. Power measurements

In the case of the FPGA, the power consumption is not dependent upon the frame size. Thus, it makes sense to report the result in terms of energy consumption per frame. Table 3 shows that the embedded system's power fluctuations due to the number of coefficients are negligible since only the filter IP core is modified. It is then more meaningful to consider the power of the FIR Filter core which does vary according to N (number of coefficients).

Device static power does depend exclusively on the device size and operating temperature, called 'device static power' [11]. It is consumed by the device when it is powered up and without programming the user logic. For the XCVFX20 device, it amounts to 166 mW (all 3 voltage rails), at 25 ºC. If the power results are to be meaningful across different devices, this quantity must be considered as an offset that will vary across devices.

Table 3. Embedded system Power consumption (Watts) on the XCVFX20-11FF672 Virtex-4 FPGA

	P_{rows}	P_{cols}	$P_{reconfig-row}$	$P_{reconfig-col}$
Mean	1.2410	1.2472	1.3035	1.3097
Std	0.0059	0.0140	0.0059	0.0140

In the case of the GPU, we found that on average, it consumes 96.8, 92.5, and 88 Watts for VGA, CIF, and QCIF frame sizes respectively. Variations for different number of coefficients are negligible (around 0.1 W) since the algorithm uses the maximum number of cores regardless of the number of coefficients of the filter. The power fluctuations for different frame sizes are due to the fact that for smaller frame sizes, the GPU is moving data over a longer period of time than when it is processing.

5.4. Energy, Performance, and accuracy results

For comparing energy consumption, we only consider the energy spent by the filtering process. Thus, for the FPGA, we consider the energy consumed by the FIR filter and the ICAP cores. For the GPUs, we will also consider the energy spent during actual video processing (Equation. 6).

Figure 5 shows the energy per frame, performance (achieved frames per seconds) and accuracy results. Note that in the case of performance, we report the mean fps with its standard deviation for a given frame size. We observe an energy dependence on the number of coefficients in the FPGA case, although it is more pronounced in the GPU case. In addition, the performance dependence on the number of coefficients is negligible in the FPGA case, but noticeable in the GPU case.

In terms of PSNR (dB), the GPU gives better results due to its use of double precision. However, there is no significant difference at the output except for N = 32. In this case, we have very high PSNR values that exceed 80dB.

In terms of performance, the GPU always prevails due to the massive amount of parallelization achieved in the algorithm coupled with the high operating frequencies. The speed up (GPU over FPGA) is about 9X, 5X, and 3.3X for VGA, CIF, and QCIF frame sizes respectively. For smaller frame sizes, the time consumed in allocations and transfers is closer to the processing times.

In terms of energy per frame, the FPGA implementation is much better than the GPU. The GPU implementation consumes 6, 9, and 19 times more energy than the FPGA's for VGA, CIF, and QCIF frame sizes respectively.

Our results suggest that the FPGA implementation provides a low-energy solution at near real-time performance. Here, we refer to frame rates that are over 30 fps as achieving real-time performance. On the other hand, when energy consumption is not an issue, the GPU implementation is superior, delivering much higher performance at slightly better accuracy.

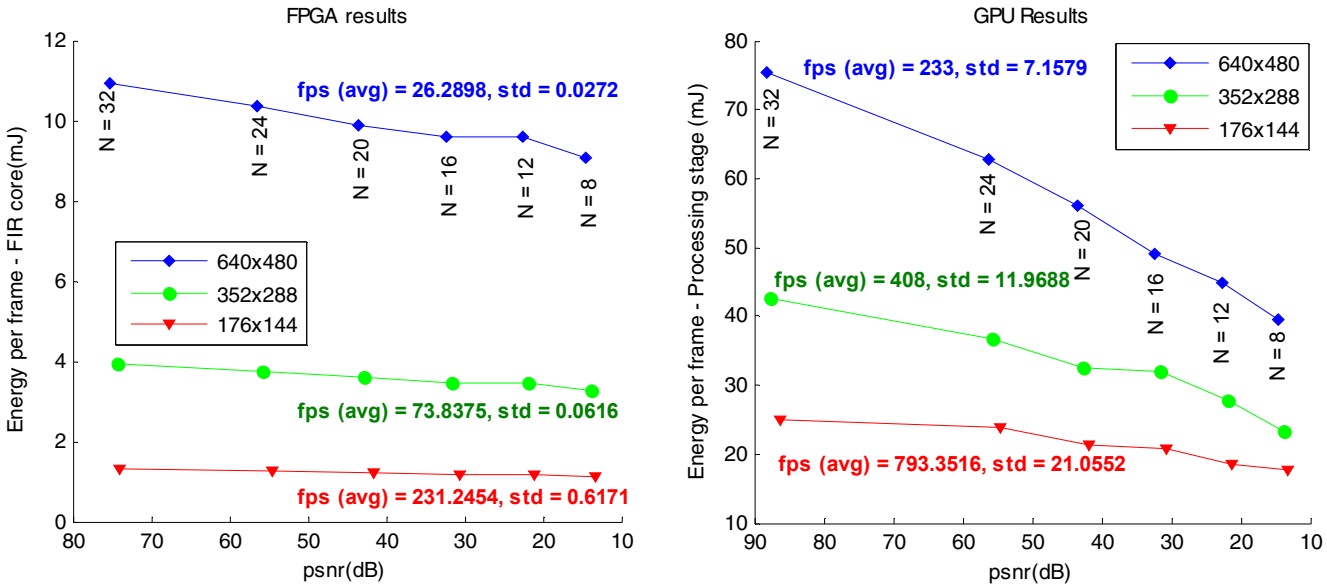

Figure 5. Performance, energy, and accuracy results for both FPGA and GPU. N: number of coefficients

6. CONCLUSIONS

This work successfully compares energy, performance (in frames per second), and accuracy for both FPGA and GPU implementations. Moreover, these 2 implementations allow the user to modify the 2D FIR Filter at run-time. The results indicate that separable 2D FIR filtering implementations can deliver excellent accuracy for both the FPGAs and the GPUs. However, based on energy consumption, FPGAs are preferred for low-energy applications. On the other hand, GPUs should be considered for high-performance, high-power (high-energy) applications.

7. REFERENCES

[1] Alan Bovik ed., Handbook of Image and Video Processing. Academic Press, 1st Edition, May 2000.

[2] D. Llamocca, M. Pattichis, and A. Vera, "A Dynamically Reconfigurable Parallel Pixel Processing System", in *Proceedings of the International Conference on Field Programmable Logic and Applications FPL'2009*, Prague, Czech Republic, Sep. 2009.

[3] D. Llamocca, M. Pattichis, and G. Alonzo Vera, "Partial Reconfigurable FIR Filtering System using Distributed Arithmetic", *International Journal of Reconfigurable Computing*, vol. 2010, Article ID 357978, 14 pages, 2010.

[4] D. Llamocca, M. Pattichis, "Real-time dynamically reconfigurable 2-D filterbanks", in *Proceedings of the 2010 IEEE Southwest Symposium on Image Analysis & Interpretation*, Austin, TX, May. 2010.

[5] "Two-dimensional Linear Filtering (XAPP933) by Robert Turney", v1.1 ed., Xilinx Inc., 2100 Logic Drive, San Jose, CA, 95124, Oct. 2007.

[6] S. Collange, D. Defour, A. Tisserand, "Power Consumption of GPUs from a Software Perspective", in *Proceedings of the 9th International Conference on Computational Science (ICCS'09)*, pp.914-923, Springer, 2009.

[7] V. Podlozhnyuk, "Image Convolution with CUDA", NVIDIA, June 2007.

[8] Cope, B., Cheung, P.Y.K., Luk, W., Witt, S., "Have GPUs made FPGAs redundant in the field of video processing?", in *Proceedings of the 2005 IEEE International Conference on Field Programmable Technology*, pp. 111-118, Singapore, Dec. 2005.

[9] Jones, D.H., Powell, A., Bouganis, C.-S., Cheung, P.Y.K., "GPU versus FPGA for High Productivity Computing", in *Proceedings of the International Conference on Field Programmable Logic and Applications FPL'2010*, Milan, Italy, Sep.2010.

[10] CUDA C Programming Guide, NVIDIA, v 3.2, Sept. 2010.

[11] Power Methodology Guide (UG786), Xilinx, San Jose, CA, v13.1 edition, March 2011.

[12] G.A. Vera, "*A dynamic arithmetic architecture: precision, power, and performance considerations*", Ph.D. Dissertation, University of New Mexico, Albuquerque, NM, USA, May 2008.

[13] C. Claus et al, "A multi-platform controller allowing for maximum dynamic partial reconfiguration throughput", *in Proceedings of the International Conference on Field Programmable Logic and Applications FPL'2008*, pp. 535-538, Heidelberg, Germany, Sept. 2008.