Architecture Exploration for Energy-Efficient Embedded Vision Applications: From General Purpose Processor to Domain Specific Accelerator

Maria Malik[1], Farnoud Farahmand[1], Paul Otto[1], Nima Akhlaghi[1], Tinoosh Mohsenin[3], Siddhartha Sikdar[2], Houman Homayoun[1]

[1]Department of Electrical and Computer Engineering, [2]Department of Bioengineering,

George Mason University, Fairfax County, {mmalik9, ffarahma, potto, nakhlagh, ssikdar, hhomayou}@gmu.edu

[3]Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, {tinoosh}@umbc.edu

Abstract- OpenCV applications are computationally intensive tasks among computer vision algorithms. The demand for low power yet high performance real-time processing of OpenCV embedded vision applications have led to developing their customized implementations on state-of-the-art embedded processing platforms. Given the industry move to heterogeneous platforms which integrates single core or multicore CPU with on-chip FPGA accelerators and GPU accelerators, the question of what platform and what implementation, whether hardware or software, is best suited for energy-efficient processing of this class of applications is becoming important. In this paper, we seek to answer this question through a detailed hardware and software implementation of OpenCV applications and methodically measurement and comprehensive analysis of their power and performance on state-of-the-art heterogeneous embedded processing platforms. The results show that in addition to application behavior, the size of image is an important factor in deciding the efficient platform in terms of highest energy-efficiency (EDP) among hardware accelerators on FPGA and software accelerators on GPU and multicore CPUs. While hardware implementation on ZYNQ shown to be the most performance and energy-efficient for image size of 500x500 or less, software GPU implementation found to be the most efficient and achieves highest speedup for larger image sizes. In addition, while for compute intensive vision applications the gap between FPGA, CPU and GPU reduces as the size of image increases, for non-intensive applications, a large performance and EDP gap is observed between the studied platforms, as the size of the image increases.

Keywords-Computer vision; OpenCV; GPU; FPGA; Multicore CPU

I. INTRODUCTION

Recent innovations in the semiconductor industry made it possible to integrate various sensors and computing components in an embedded system on a chip (SoC) processing platform. Mobile platforms use embedded SoC to process sophisticated and computationally intensive computer vision applications. An example of such system is a wearable glass with camera, which has numerous applications in healthcare, robotics, navigation and security[1, 2].

Low power yet high performance image processing on embedded vision platform has many applications in various domains including healthcare, security, telecomm and IoT, just to name a few. An example is in healthcare, for patient rehabilitation, where a user's movement or posture needs to be accurately tracked to detect the need for corrective action[3]. This is true if a patient is at risk of falling and an alert needs to be automatically generated in case of injury. Another example is in remote sensing and monitoring, where accurate tracking required for autonomous drones is partially performed by interpreting location change through image content being received and processed at a high frame rate [4,5]. For instance, Parrot's AR.Drone- a dual camera system whose vertical camera generates images at 60 frames a second rely on a navigation system that uses corner detection algorithms that run Sobel filtering on large images at high frame rate[4]. In general, remote sensing and monitoring applications rely on sophisticated computer vision algorithms to run on low power embedded hardware to maximize their operating time. Similarly in the field of security, image processing at the camera before transmission is critical to reduce bandwidth of distributed surveillance systems[6]. These systems may employ filtering on the embedded camera hardware and only transmit the approximated target state coefficients[6]. This eliminates the need for a high bandwidth links, while allowing multiple target information to be captured.

While demand for high performance computing vision continues to grow, the physical design constraints, such as power and density, have become the dominant limiting factors for scaling out embedded computing systems. Current processor design, based on commodity homogeneous processors, are not the most efficient in terms of performance/watt to process compute intensive applications[15, 19, 20]. To address the energy-efficiency challenge, heterogeneous architectures have emerged as a promising solutions in high performance as well as embedded systems to significantly improve the energy-efficiency by allowing applications to run on a computing core that matches the resource needs more closely than a single one-size-fits-all general purpose core. A heterogeneous chip architecture integrates cores with various micro-architectures (in-order or out-of-order) or instruction set architectures (Thumb and x86) with on-chip GPU or FPGA accelerators to provide more opportunities for efficient workload mapping so that the application can find a better match among various components to improve power efficiency. In particular, hardware acceleration through specialization, which is enabled by tight integration of CPU core and FPGA logic, has received renewed interest in recent years, partially in response to the dark silicon challenge. Examples of heterogeneous architectures in embedded domains are Xilinx ZYNQ (CPU+FPGA), NVIDIA Tegra (CPU+GPU), Qualcomm Snapdragon (CPU+DSP+GPU) and Samsung Exynos (Big +Little CPU+GPU). Given the diversity of architectures for these emerging heterogeneous platforms, the question is which

978-1-4673-9039-2/16 $31.00 © 2016 IEEE
DOI 10.1109/ISVLSI.2016.112

IEEE computer society

architecture best suits the power and performance requirement of computer vision applications becomes important.

The objective of this paper is to answer this question through a detailed hardware and software implementation of various OpenCV applications and methodically measurement and comprehensive analysis of their power and performance on state-of-the-art heterogeneous embedded processing platforms. Among OpenCV applications, we are mainly focusing on the convolution based filters (Sobel and Gaussian) representing more computationally intensive vision applications, as well as general image processing techniques (Average subtraction, Image thresholding, and Image scaling) representing less computationally intensive vision applications. Filtering algorithms are being used extensively in various vision domains for feature detection, image analysis and noise reduction [7–9]. The Sobel filter is employed in application involves with edge detection[10, 12, 18, 22] where Gaussian filter is utilized for noise reduction and suppressing image details [13]. These filters usually are implemented by using convolution, where an image is convolved with the kernel corresponding to a particular filter. Convolution is a computationally intensive operation mainly for real-time performances; therefore there is a need for better optimization at the system and algorithm level to enhance the power efficiency.

For the choice of heterogeneous architecture, our experimental work implemented these vision algorithms on Nvidia Tegra, Xilinx ZYNQ and Multicore Intel ATOM and ARM to study the choice between Multicore CPU with diverse ISA (x86 ATOM vs ARM thumb), GPU, as well as FPGA implementations. To find out how the results are sensitive not only to application behavior (computationally intensive vs non-intensive) but also image characteristics, we measure and analyze performance and power consumption in terms of energy-delay product (EDP) for several image sizes.

Several research works have reported the performance results of parallel implementation of computer vision algorithms on CPU and compared it with the accelerator implementations [11, 16, 24]. Cope et al, have compared the implementation performance of image convolution on GPU, FPGA and CPU [12]; Russo et al, have compared image convolution processing on GPU and FPGA [23]; Also Asano et al have investigated the performance comparison of two-dimensional filter on FPGA, GPU and CPU[17, 25]; however none of this work has studied the trade-off between power and performance on state-of-the-art embedded heterogeneous platforms. To the best of our knowledge this is the first experimental work that compares hardware and software implementations of several OpenCV computer vision kernels on state-of-the-art heterogeneous embedded platforms.

The rest of the paper is organized as follows: In section II, we explain the experimental methodology and setup. In section III, we present the results. Section IV discusses in details the power and performance measurements results across studied architectures. Finally, in section V we present the conclusion remarks.

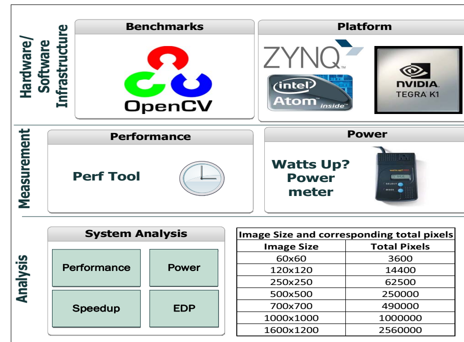

Figure 1: Methodology

II. METHODOLOGY

Two convolution based filters (Sobel and Gaussian) and three general image processing techniques (Average subtraction, Image thresholding, and Image scaling) were implemented on three different platforms: GPU, FPGA and two types of CPU (ATOM and ARM). The algorithms are implemented as individual standalone programs. Multicore implementation of studied applications are performed with OpenCV and OpenMP.

The algorithms processed seven different images sizes in the range of 60 by 60 up to 1600 by 1200. The total pixels calculated by multiplying the image dimensions and is used

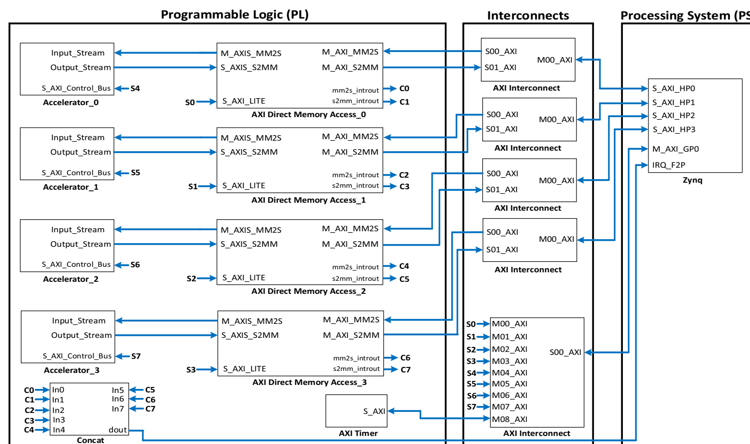

Figure. 2: FPGA Implementation Design

when comparing algorithm results using a pixel processing per time unit. The methodology on which experiments are conducted is presented in the Figure 1.

To investigate the effect of optimization at the hardware and software levels (application software tuning), several different hardware optimization and software tuning sceneries were implemented. For instance, the CPU results for different compiler optimization levels from O0 (no optimization) to O3 (most optimized one) on single core and multi-core were compared with the results of the other two hardware platforms (GPU and FPGA), which considered to be a better optimized platforms (performance and EDP wise). Image tiling were also implemented as a program tuning to optimize the process in parallel manner. Next we will discuss in more details the implementation details of the studied OpenCV applications on each platform.

A. GPU implementation

The GPU implementation uses an NVIDIA Jetson TK1 developer kit which is built around the Tegra K1 processor. The processor is composed of a GPU and multicore CPU. The GPU has 192 NVIDIA CUDA cores while the CPU is a "4-Plus-1" 2.32 GHz ARM quad-core Cortex-A15. A single core of the ARM processor speed was fixed at 1.224 GHz with the other cores turned off. The GPU speed was fixed at 852 MHz. We run each code module as a standalone program by a script file that accepts the program configuration information as command line arguments. Each program had an internal loop to run the OpenCV code multiple iterations which provided the execution time per iteration for the study. The image load time was excluded from each run. Timing was performed using the C API clock() function to be consistent with the ARM timing.

B. FPGA implementation

For FPGA implementation, we use Xilinx ZEDBOARD with a Zynq-7020 SoC containing of dual-core ARM and a FPGA. The hardware design consists of 3 major parts shown in Figure 2:

a) *Processing system (PS):* ARM processor inside Zynq Platform.

b) *Interconnects:* The high performance (HP) ports are used for transactions between PS and Programmable logic (PL). The data (image) is transferred to/from the accelerator in PL using ARM core. AXI Interconnects IPs generated automatically using Vivado design suit take care of data transaction between ARM PS and PL in memory mapped mode.

c) *Programmable logic (PL)* which has 3 major IPs:

i. *Accelerators*: These are the main IPs which are generated using Vivado HLS. We developed HLS (High Level Synthesis)_ready C++ code by the Vivado HLS OpenCV library which is provided by Xilinx, specifically for each application. The HLS_ready C++ code contains different pragmas and tweaks in comparison with regular C++ code. We have implemented the C++ code because all C++ coding capabilities are not supported by HLS process (e.g., Dynamic memory allocation, structures and etc.). In the next step HLS_ready C++ code is translated to VHDL and the corresponding IP is generated using Vivado HLS. Two different interfaces are used for each accelerator IP, AXI4_Lite interface for sending control data such as image size and AXI4_Stream video interface for sending and receiving input and output image respectively. The reason for this selection is that AXI4_Lite doesn't have burst transaction capability and it is suitable for sending small amount of data with specific address. However, AXI_Stream let you send and receive data in burst mode, so it will give you much better performance for transferring large amount of data without addresses. Coding Hierarchy: At first we convert the input image from AXI4_Stream format to HLS Mat format using hls::AXIvideo2Mat. We then process the corresponding image in Mat format and at the end the output image is converted from Mat format to AXI4_Stream format using hls::Mat2AXIvideo. Therefore we can transfer the output result using AXI DMA.

ii. *AXI Direct Memory Access (DMA)*: This IP is added to block design from Xilinx IP catalog and it converts the stream transaction to memory map and let the user to write and read to and from DDR memory.

iii. AXI Timer: Added from Xilinx IP catalog which measure the execution time of hardware implementation.

Table 1: Architectural Parameters

Processor	**Intel ATOM C2758**	**ARM Cortex-A15**
Max. Operating Frequency	2.40 GHz	2.32 GHz
Micro-architecture	Silvermont	ARMv6
L1I Cache	32 KB	32 KB
L1D Cache	24 KB	32 KB
L2 Cache	4×1 MB	2 MB
System Main Memory	8 GB	2 GB
In-order/Out-of-order	Out-of-order	Out-of-order
Word Width	64 bits	32 bits

Our implementation consists of four DMA blocks, each of them connected to one of the HP ports in the PS. Each of these DMAs connected to one accelerator IP and take care of data transfer between accelerator and processor. As a result, we can process four images in parallel at the same time. Therefore, the idea is to split the image into four smaller chunks and transfer each of them separately using one of the four HP ports to accelerator IPs. The received data are then merged to build the complete image. With this method we can reduce the execution time by a factor of four. All execution time results include the data transfer time and they are based on the maximum frequency i.e. 84.9MHz, 68.7MHz, 76.3MHz, 64.1MHz and 51.3MHz for Image threshold, Image scaling, Average Subtraction, Sobel and Gaussian, respectively.

C. CPU implementation:

We conducted our study using both Intel Atom and ARM CPU. The Intel ATOM C2758 has four active processing cores and two levels of cache hierarchy. The processor hosted the Ubuntu 13.10 operating system with a Linux 3.11 kernel. The ARM architecture is NVIDIA's "4-Plus-1" 2.32GHz ARM quad-core Cortex-A15 CPU. Both platforms have four active processing cores, therefore application multithreading up to four parallel threads is enabled. The NVIDIA version also uses Ubuntu Linux as its OS. Table 1 summarizes the key architectural parameters of the microservers. We use Perf to capture the performance characteristics of the studied

applications on ATOM. Perf exploits Performance Monitoring Unit (PMU) in the processor to measure performance as well as other hardware events accurately. Because this tool is not available for monitoring the ARM on the Tegra processor, the clock() function of the C API was used. This function factors in if multicore processing is used to execute the code. For measuring power dissipation of the microserver, Wattsup PRO power meter is used. Wattsup power meter measures and records power consumption at one second granularity. The power reading is for the entire system, including core, cache, main memory, hard disks and on-chip communication buses. We have collected the average power consumption of the studied applications and subtracted the system idle power to calculate the dynamic power dissipation of the entire system.

III. Results

To compare different platform, we have presented the performance and EDP results in this section. Additionally, the average performance and EDP results over all applications were calculated to investigate the optimal platform based on best performance and maximum energy efficiency for all applications. To compare the results in the best case scenario with GPU and FPGA, the CPU frequencies of 2.4 and 1.2 were used for execution time and EDP respectively. Also the compiler software optimization was set to -O3 (the most optimized case).

A. Single core and Multi-core CPU implementation

The results of the single-core and multi-core implementation of all studied applications on CPU platform (ATOM and ARM) with the biggest image size are compared in Table 2. Both ATOM and ARM demonstrate that for all the applications multi-core provides better performance compared to the single core but they clearly needs more power. Interestingly, for energy-efficiency (EDP), multicore results outperform single core in both CPU platforms. Comparing ATOM and ARM results, ATOM provides better performance than ARM and in terms of energy-efficiency, in most cases it results in the lowest energy-efficiency. Considering multicore, as compared to the single core, provides better performance and EDP, the rest of the paper compares the multi-core CPU implementation with FPGA and GPU.

B. Performance Analysis

Figure 3 illustrates the speedup achieved on FPGA, GPU and ATOM architecture compared to the ARM (considered as a baseline) for all the studied applications. Image threshold and Image scaling results illustrate that FPGA has the highest speedup for small images (60x60 and 120x120) and GPU

Table 2: Single core and Multi core CPU Results

		ARM		ATOM	
		Single	Multi	Single	Multi
Image Thresholding	Exe(msec)	2.01	1.25	0.901	0.708
	EDP(Jsec)	1.29E-05	3.68E-06	2.27E-06	1.95E-06
Image Scaling	Exe(msec)	40.47	23.65	29.09	14.56
	EDP(Jsec)	3.60E-03	9.54E-05	4.13E-04	2.12E-04
Average Subtraction	Exe(msec)	8.48	6.38	16.6	8.35
	EDP(Jsec)	1.80E-04	1.20E-04	1.20E-03	6.60E-04
Gaussian Blur Filter	Exe(msec)	23.93	11.43	19.01	9.67
	EDP(Jsec)	1.45E-03	1.35E-03	2.44E-03	9.70E-04
Sobel Filter	Exe(msec)	150	75.3	131.079	66.81
	EDP(Jsec)	4.90E-02	3.10E-04	5.21E-04	2.86E-04

provides maximum speedup for larger images (250x250-1600x1200). Image average subtraction shows that FPGA shows the largest speedup for image size up to 500x500 and GPU has dominated at the larger image sizes (700x700-1600x1200). The result for Sobel filtering illustrates that FPGA has attained the highest speedup on all the image sizes except for the largest image size where GPU has better performance. In Gaussian blur, FPGA has achieved the highest speedup only at the smallest image size. ATOM shows highest speedup for image sizes ranges from 120x120 to 500x500 and GPU provides the maximum speedup for the larger image sizes. In sum, with image sizes larger than 500x500, GPU is the winner in terms of speedup. Comparing the ATOM and ARM results, ATOM outperforms ARM for all the image sizes expect at the smaller image size.

C. Energy Efficiency Analysis

In order to characterize the energy efficiency, we evaluate Energy Product Delay (EDP) metric to investigate trade-off between power and performance when running image processing applications on FPGA, GPU and multicore CPU (ATOM and ARM). Figure 4 shows the EDP of the studied architectures (A = FPGA, GPU, ATOM) compared to ARM. The power is almost constant for FPGA with values ranging 1.735-1.755 for studied applications. However, power varies significantly: 0.86-6.06, 0.5-3.512, and 0.20-2.4 for GPU, ATOM, and ARM, respectively for the applications across different image sizes.

For non-computationally intensive vision applications; image scaling, threshold and averaging, for small image size of below 250x250, FPGA is a more efficient implementation than GPU and multicore CPU. However as the size of image increases the trend quickly moves to GPU. The gap between GPU and FPGA increases as the size of image increases.

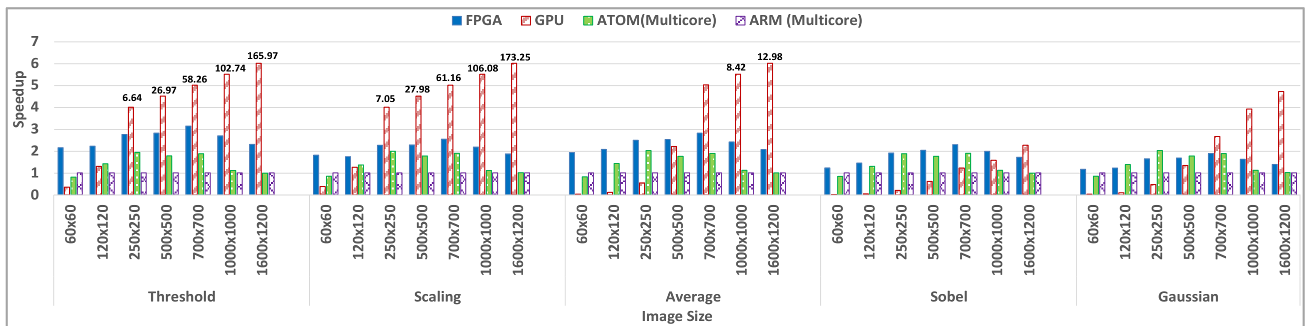

Figure 3. Speedup on studied architectures (A = FPGA/GPU/ATOM) compared to ARM of various image processing applications over several image sizes

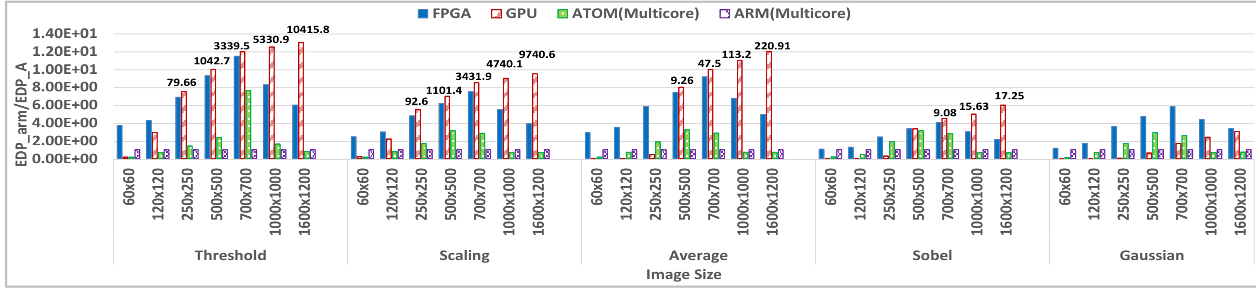

Figure 4. EDP on studied architectures (A = FPGA/GPU/ATOM) vs ARM of various image processing applications over several image sizes

However, for computationally intensive vision applications a different trend is observed. While for small image size of below 500X500 FPGA is clearly a winner in terms of energy-efficiency, for larger image sizes GPU becomes competitive with FPGA. For Gaussian filter, FPGA is always the efficient platform compared to other platforms across all studied image sizes. Comparing ATOM and ARM, while in some cases ATOM is competitive with ARM and other platforms in terms of energy-efficiency, in most cases it results in the higher energy efficiency. This is due to the fact that ATOM ISA (X86) and machine width (64 bits) mainly designed in response to high performance demand and not necessarily low power concerns [21].

D. Average per Pixel Analysis

Figure 5a shows the results for performance and EDP over each pixel across different platforms. Figure 5b demonstrates the performance and EDP over each pixel across different applications. EDP results demonstrates that FPGA has the best result in case of small images for both computational intensive and non-intensive applications. However, GPU is the best platform for processing large images. As we can observe, a similar trend is seen for application execution time.

IV. DISCUSSION

The GPU results show that its EDP value is almost insensitive to image size by showing only a small increasing trend. This is due to the overhead for processing small images and the speedup advantage at large image sizes. Further the power consumption was nearly constant during the code execution. These factors lead to a near constant EDP value. It is expected that if the image size continues to increase then the EDP would continue its slow increase and its overall advantage compared to other platforms would grow since the EDP slope rates differ.

Additionally, for the GPU its EDP advantage over the other processing technologies was much greater for image thresholding, image scaling, and average subtraction as compared to Sobel or Gaussian filtering for image sizes greater than 250 by 250. It is only at the smaller image sizes that the FPGA outperforms the GPU in terms of EDP. This is due to the image thresholding, image scaling, and average subtraction exploiting the GPUs shared memory layout and SIMD design. Because these are scalar operations on an array of values each GPU thread can process the values in parallel. The FPGA must synthesize this hardware which is not as efficient as the bare metal GPU design. Additionally, the Sobel and Gaussian filters had additional processing steps or a combination of multiple gather/scatters which can be directly synthesized in the FPGA. Further, the Sobel filter requires a square root operator which is not as efficiently performed in the GPU. All of these factors lead to the GPU's performance difference.

The ARM processor performed consistently well, especially at small image sizes. Its performance often exceeded the

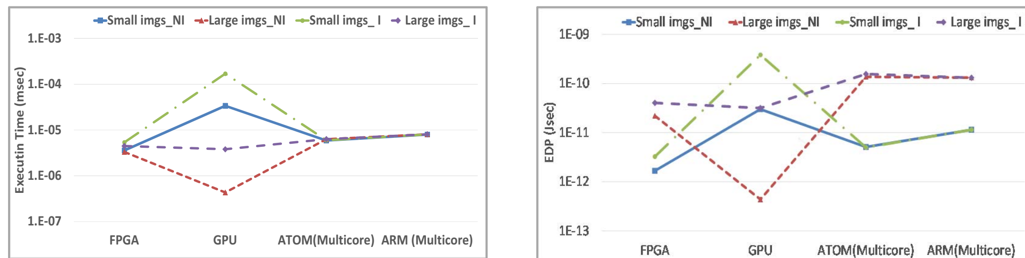

Figure 5(a). Per pixel performance and EDP across different platforms

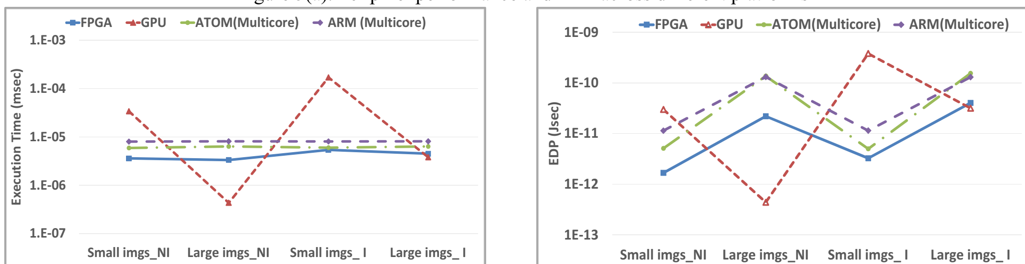

Figure 5(b). Per pixel performance and EDP across different applications (Non-intensive – NI- and intensive – I-)

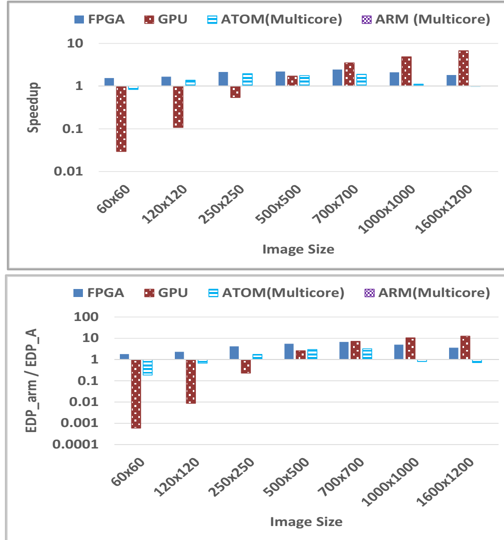

Figure 6. Speedup and EDP of studied architectures (A = FPGA/GPU/ATOM) compared to ARM averaged across applications over several image sizes

performance of the GPU and ATOM, while just trailing the FPGA. This in part is due to the ARM's efficient instruction set, however at higher image sizes the GPU is able to more efficiently process the larger images.

The results for the FPGA show that its overall EDP performance is dependent on the image size being processed. This is due to its varying execution time with nearly constant power consumption. As can be seen in the execution time figure the overall trend is that at very low image sizes the FPGA performs the best and then as the image size increases its execution time is become comparable with the ARM multicore. In case of ARM and ATOM multicore implementation, execution time and EDP result increases almost exponentially for bigger size images, due to sequential execution characteristic of CPU in compared with GPU and FPGA. Figure 6 shows the speedup and EDP results averaged across over all the studied applications on various architectures (A = FPGA/GPU/ATOM) compared to ARM. We can observe, on average across all studied applications FPGA has the best results for image sizes smaller than 700x700 in case of execution time. In case of EDP, GPU is the most efficient platform for image sizes larger than 500x500 while FPGA yields the best EDP for smaller image sizes.

V. CONCLUSIONS

Low power yet high performance image processing on embedded vision platform has many application in various domains including healthcare, security, telecomm and IoT, just to name a few. Heterogeneous architectures combining on chip accelerator such as FPGA and GPUs with multicore general purpose CPUs are emerging as promising solutions to significantly improve the energy-efficiency of this class of applications. Therefore, the question of which of these architectures provide the best power and performance results for computer vision applications becomes important. Our experimental results across a number of OpenCV applications demonstrate that for compute-intensive applications such as Gaussian blur and Sobel filter, FPGA achieves highest performance for image sizes smaller than 500X500 while GPU is the winner for larger images, compared to other platforms. The similar trend is observed for energy efficiency, the results demonstrate that FPGA has the lowest EDP for small image sizes and GPU for the bigger images as EDP remains almost unchanged for GPU across various image sizes. Overall, the large performance and EDP gap is observed between hardware implementation on FPGA and software implementation on GPU and multicore CPU across various OpenCV applications and different image sizes.

VI. ACKNOWLEDGMENTS

This work was supported in parts by the National Science Foundation under grant CSR-1526913 and CPS-1329829.

REFERENCES

[1] B. Gillette, “Hospital tests Google Glass with dermatology patients,” Dermatology Times E-News, 19-Mar-2014. [Online].

[2] O. J. Muensterer, et al., “Google Glass in pediatric surgery: An exploratory study,” Int. J. Surg., vol. 12, no. 4, Apr. 2014.

[3] H. Zhou et al.“Human motion tracking for rehabilitation-A survey,” Biomed. Signal Process. Control, Jan. 2008.

[4] B. Pierre-Jean, “The Navigation and Control Technology Inside the AR.Drone Micro UAV,” 2011, pp. 1477–1484.

[5] F. de B. Martins, et al., “Visual-Inertial Based Autonomous Navigation,” in Robot 2015:

[6] S. Fleck, et al., “Adaptive Probabilistic Tracking Embedded in Smart Cameras for Distributed Surveillance in a 3D Model,” EURASIP JES 2007.

[7] Z. Li, et al., “Adaptive nonlocal means filtering based on local noise level for CT denoising,” Med. Phys., vol. 41, p. 011908, Jan. 2014.

[8] E. S. L. Gastal, “Efficient high-dimensional filtering for image and video processing,” 2015.

[9] Q. Gao, Y. Zou, J. Zhang, S. Liu, Z. Xie, and S. Chen, “Missile vision guidance based-on adaptive image filtering,” in IEEE ICIA 2015.

[10] P.I. India, et al.,“Crack Detection of Medical Bone Image Using Contrast Stretching Algorithm with the Help of Edge Detection,” IJSET 2015.

[11] M. K. Tavana, et al., “Energy-efficient mapping of biomedical applications on domain-specific accelerator under process variation”, in proc. of ISLPED 2014

[12] F. A. Hussin, et al.,“Optimization of Processor Architecture for Sobel Real-Time Edge Detection Using FPGA,” in. IRECOS 2013.

[13] F. Qin, et al., “Blind Single-Image Super Resolution Reconstruction with Gaussian Blur and Pepper & Salt Noise,” J. Comput., 2014.

[14] B. Cope, “Implementation of 2D Convolution on FPGA, GPU and CPU,” Dept. of EEE, Imperial College London.

[15] M. Malik, et.al.,"Big data on low power cores: Are low power embedded processors a good fit for the big data workloads?." in proc. of ICCD 2015

[16] Page et al.,"Low-Power ManyCore Accelerator for Personalized Biomedical Applications", in proc. of GLSVLSI 2016

[17] S. Asano, et al., “Performance comparison of FPGA, GPU and CPU in image processing,” in FPL 2009.

[18] M. Chouchene, et al., “Efficient implementation of Sobel edge detection algorithm on CPU, GPU and FPGA,” IJAMC, Jan. 2014.

[19] M. Malik, et. al., “System and Architecture Level Characterization of Big Data Applications on Big and Little Core Server Architectures,” presented at IEEE Big Data 2015

[20] Gutierrez, A. etal. "Integrated 3D-stacked server designs for increasing physical density of key-value stores." Proc. of ASPLOS, 2014

[21] Blem, Emily, et. al. "Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architecures" HPCA 2013

[22] P. Otto, et al. "Power and Performance Characterization, Analysis and Tuning for Energy-efficient Edge Detection on Atom and ARM based Platforms.", In proc. of ICCD 2015

[23] L. M. Russo, et al., “Image convolution processing: A GPU versus FPGA comparison,” in SPL 2012.

[24] J. Bisasky et al., "A Many-core Platform for Biomedical Signal and Image Processing" in proc. of ISQED 2013

[25] T. Saegusa, T. Maruyama, and Y. Yamaguchi, “How fast is an FPGA in image processing?,” in FPL 2008