Energy-Efficient Median Filter on FPGA

Andrea Sanny
Ming Hsieh Dept. of Electrical Engineering
University of Southern California
Los Angeles, CA 90089
Email: sanny@usc.edu

Viktor K. Prasanna
Ming Hsieh Dept. of Electrical Engineering
University of Southern California
Los Angeles, CA 90089
Email: prasanna@usc.edu

***Abstract*—Median filters are a popular method for noise extraction, with much work done in the community to achieve high throughput and low hardware cost. In contrast, energy efficiency remains an untapped area for improvement though it has become a topic of increasing interest. We deduce memory to be the main contributing factor through energy consumption analysis of our median filter architecture. To lower memory energy demands, we use a memory activation scheduling technique, developing an optimal schedule for memory blocks and enabling the minimum number of blocks required each cycle, while deactivating the other blocks. We implement our median filter architecture on a state-of-the-art FPGA to evaluate the performance, using image sizes from 128×128 to 1024×1024 pixels with standard pixel depths of 8 bpp, 16 bpp, 24 bpp and 32 bpp. Our implementation has up to 53% of the peak performance of the target device. The post place-and-route results show that our energy-optimized median filter implementation has an average of $4.0\times$ higher energy efficiency in comparison with the baseline architecture with fully-enabled memory and can maintain at least 400 frames/s for a 512×512 image for any pixel depth.**

I. Introduction

FPGAs are well-known for their flexible, reconfigurable nature and strong suitability as a platform for computationally-intensive algorithms [9]. Programmed specifically for a selected problem, they can achieve higher performance with lower power consumption than general-purpose processors and are a promising implementation technology for applications including signal, image and network processing tasks.

One of the most common issues that arise in image processing is the addition of noise, which can be caused by errors in data transmission or malfunctioning pixels in sensors. A commonly-used remedy for smoothing out noise is applying a median filter (a nonlinear digital filter). Unlike linear filtering, median filtering can reduce high frequency and impulsive noise without resulting in adverse effects to the edge information of the image, improving the quality of the result.

Due to the popularity of median filtering, a variety of methods have been developed, with one of the most prominent being the use of sorting to find the median of the pixels [1], [3]. These sorting-based methods focus on minimizing the needed hardware resources and achieving optimizations to improve the operating frequency. Though these are important metrics to consider, an additional metric becoming more prevalent is energy efficiency. With the desire for low-power devices, energy efficiency must be addressed; yet to our knowledge there has not been a focus on developing an energy-efficient method.

In this work, we use a sorting-based method for our median filter with the image being stored in on-chip memory, exploring the design space of the median filter. By analyzing the main components' effects on energy efficiency, lowering memory energy is deduced to be the key to creating a more energy-efficient design. We propose the solution of a memory scheduling technique, which uses an activation schedule on the blocks of memory, attempting to enable only the minimum number of needed blocks to lower energy dissipation. Next, we estimate the peak energy efficiency for a median filter, which is the upper bound of energy efficiency for any median filter method on the target device. We compare our median filter design against this bound to demonstrate the improved performance of our implementation over the baseline.

Using the post place-and-route results on a state-of-the-art device, we determine the extent of improvement of energy requirements, while maintaining an adequate solution to median filtering with regards to frames per second. We vary the image size and number of bits per pixel to evaluate the amount of possible savings of energy. The main contributions of this work are summarized below:

- An evaluation of potential energy hot spots within the median filter components (Section III-B)
- A memory activation scheduling technique for an energy-efficient approach to median filtering (Section III-C)
- An optimized architecture which can sustain up to 53% of the peak performance of the target device (Section IV)
- Development of an upper bound on the peak performance of any median filtering algorithm on the selected platform (Section IV-A)
- A performance comparison and evaluation of our proposed technique against the baseline with respect to throughput and energy efficiency (Section IV-B)

The paper is organized into the following sections, beginning with Section II, which describes the background and related work. Section III presents the median filter architecture, power analysis and proposed memory activation scheduling technique. Section IV gives an overview of the experiments and the resulting analysis. Section V concludes the paper.

This work has been funded by DARPA under the grant number HR0011-12-2-0023.

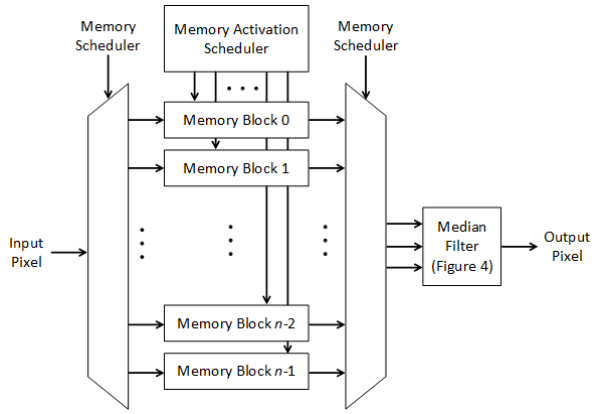


Fig. 1: High-level organization of our approach

II. Background and Related Work

A. Background

Median filtering is used mainly for image restoration and smoothing, to remove undesirable noise that affects image quality while preserving the image features. Originally suggested by Tukey [13] as a one-dimensional waveform, it has been widely used to smooth out images affected by spiky noise distributions as well as evolving into an applicable method for two-dimensional images. An image is stored as a 2-D matrix of pixel values, and each pixel of the input image uses a $k \times k$ window of surrounding pixels used to determine the median value of those pixels. Each pixel is represented by a number, specifying the gradient or color. By selecting the median, the filter can remove the extreme values beyond the normal distribution of its surrounding pixels caused by noise and as well as attempt to determine the most accurate pixel value within the window.

Generally, a 3×3 window is implemented in proposed solutions, considered to be a common and effective size for a median filter. Too large of a window can result in blurred image features, losing the sharp clarity of the original image, while too small of a window will have little effect in filtering out noise, which defeats the purpose of a filter.

B. Related Work

There are many available solutions for attempting to achieve high throughput using median filters. One of the traditional methods is a sorting-based architecture for the median filter, using a bubble sort approach [10], [8], with an array of 32 dual-input comparators. It was improved upon by decreasing the number of required comparators by sorting only as much as necessary to determine the median, and not sorting the entire $k \times k$ matrix of a window [1], [3]. Other approaches avoided the need for sorting the elements at all by using a histogram to accumulate the pixels in the kernel, modifying the contents of the histogram as the approach moves from one pixel to the next, in order to achieve $O(r)$ complexity [7]. Perreault improves the algorithm further to achieve $O(1)$ complexity while using a histogram method [11] and Fahmy used a histogram approach to achieve high-throughput for 1-D median and weighted median filters [5]. Another approach is using a bit-level rank filter, achieving low hardware complexity and achieving high operating frequency [12].

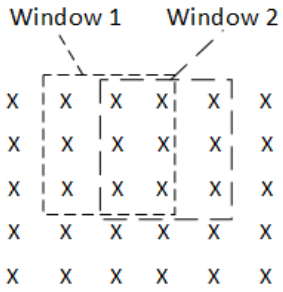


Fig. 2: Sliding window process

While these well-defined median filters rely only rank-order information of the input data given by the window, several classes of median filters have been developed in order to take temporal-order information into account as well when selecting the final pixel to be used. These include weighted median filters [2], [4], FIR-median hybrid (FMH) filters [6], and adaptive median filters [14]. However, these filters are outside the scope of this current work, and we focus instead on median filters which rely on rank-order information.

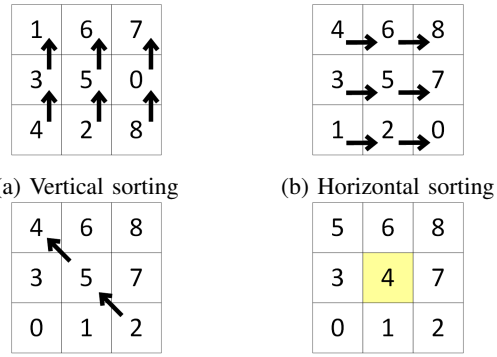
To the best of our knowledge, none of these discussed approaches have taken energy efficiency into account, focusing instead on high throughput or low hardware requirements. Images require a significant amount of memory for storage onto a target device, and there can be a high amount of additional energy spent on idle activated memory that is not being filtered. In this paper, we use a sorting-based approach due to its flexible nature, ease of pipelining, and potential for resource reduction. We show that a sorting-based approach can achieve high energy efficiency through memory activation scheduling.

III. Architecture

A. Median Filter Architecture

The overall organization is shown in Figure 1. The design consists of a memory scheduler which delegates where incoming image pixels are placed into memory as well as selecting which stored pixels are sent to be filtered at a given time. We assume that data is sent into the FPGA in row-by-row format and one pixel is read per cycle into the memory scheduler multiplexer, which is then placed into the appropriate memory block. The memory scheduler is a single unit, affecting both the inputs into and the outputs from memory. The memory activation scheduler determines which memory blocks are enabled at a given cycle. In a cycle, each enabled memory block outputs a pixel in parallel to be sent to the median filter, and the median filter outputs the median pixel for a window of pixels.

In general, median filtering uses a $k \times k$ window of pixel entries surrounding each pixel to determine the median and select that pixel as its result. We make the assumption that the filtering is done row by row, and the window can be viewed as



(a) Vertical sorting (b) Horizontal sorting

(c) Diagonal sorting (d) Result

Fig. 3: Example of median filtering using sorting

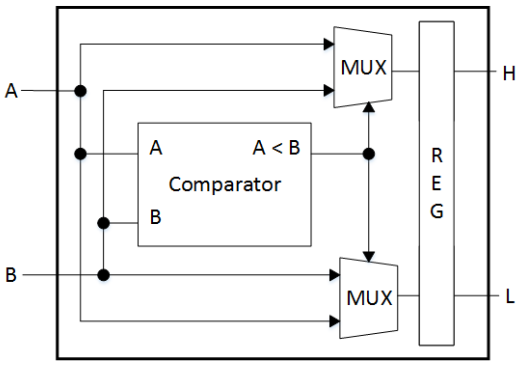


Fig. 4: Processing element

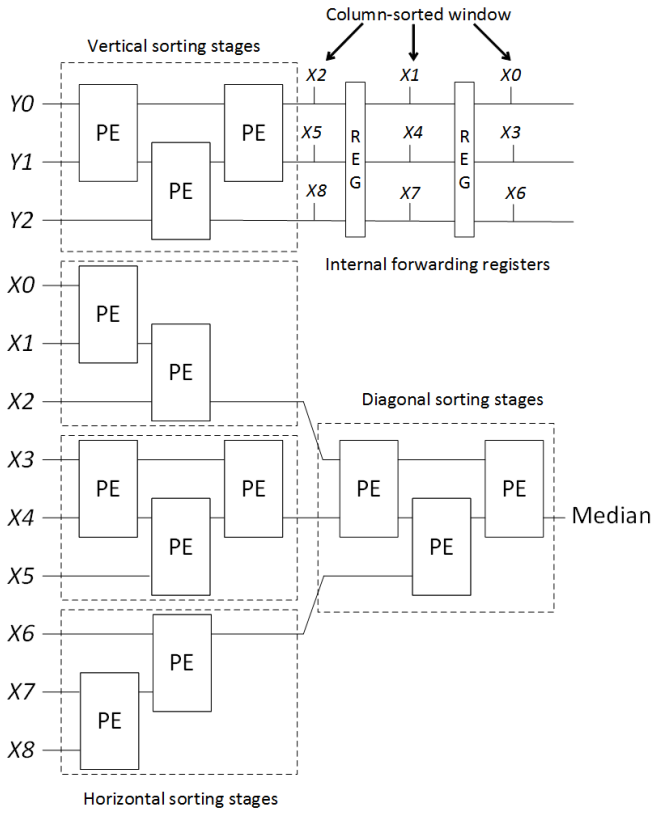


Fig. 5: 3×3 pipelined median filter architecture

a sliding window, shifting one column to the right each cycle, shown in Figure 2.

Due to the movement of the sliding window, previous columns can be retained for reuse in future windows. For a $k \times k$ sliding window, $k-1$ previous columns will be reused for the window surrounding the following pixel, which means that the sorting done on those columns need only be completed once and saved instead of resorting the columns for each new window. By saving previously sorted columns, we reduce the required number of comparisons and comparison units in our architecture. Our sorting-based approach uses the algorithm of [1] to find the median filter, with an example of the usage of the algorithm using a 3×3 window in Figure 3:

1) Sort the vertical pixels of the window, i.e, the elements within each column into ascending order

2) Sort the horizontal pixels of the window, i.e., the elements within each row in ascending order

3) Sort the cross diagonal elements, i.e., pixels that may be the median: the lowest of the largest elements, the largest of the smallest elements, and the middle of the middle elements. The final output of the window will be the middle element of the three

A basic comparison unit, or processing element (PE), used in the algorithm is defined as one two-input comparator and two two-input multiplexers, which determines the higher and lower element of two inputs. A PE is shown in Figure 4 with registered outputs for pipelining the stages. We utilized a bitonic sorting-based method and the number of comparison units required to filter a $k \times k$ window is $2k^2\lceil \log^2 k \rceil + k\lceil \log^2 k \rceil$. With the use of a sliding window, the number of comparison units for vertical sorting is reduced to $k\lceil \log^2 k \rceil$. The number of comparison units for diagonal sorting is also $k\lceil \log^2 k \rceil$. By optimizing row sorting and recognizing that two of the rows being sorted require only the least significant or most significant pixel, the number of units is $2k + (k-2)(k\lceil \log^2 k \rceil)$. Overall, the final number of comparison units is $2k + k^2\lceil \log^2 k \rceil$. Our experiments used a 3×3 window, and the median filter architecture is shown in Figure 5. Each block is a comparison unit, and by using registers in between each stage, the median filter is a highly optimized pipeline which delivers 1 pixel per cycle. $Y0 - Y2$ are input pixels from memory, which are sorted and stored into registers. These sorted pixels and the stored results form the sliding window, $X0 - X8$, which are then horizontally and diagonally sorted before determining the median.

Figure 6 shows the architecture of our median filtering approach, with a memory block being equivalent to a block RAM. For an $N \times N$ image and a $k \times k$ window, one pixel is retrieved per block RAM in parallel each cycle. Assuming each block RAM is size B, the minimum number of block

RAMs is N^2/B. To minimize the amount of block RAM required for image storage while still accessing k inputs each cycle for filtering, we use an efficient memory mapping technique. Using a scheduler, each row of incoming pixels is stored into adjacent block RAMs. For example, the first row is stored into the first block RAM, the second row into the second block RAM, and this pattern would continue until reaching the final block RAM. At this point, the memory controller returns to the first block RAM to store the current row of incoming pixels. By storing adjacent rows in separate memory blocks, the k required inputs for filtering will always be accessible for any $k \times k$ window size, assuming k is a number less the total number of block RAMs. Each cycle, k pixels are selected in parallel from the block RAMs and sent to the median filter, creating one column of the sliding window which is sorted vertically and saved into registers to be internally forwarded in a later cycle to create the window for the horizontal sorting. The internal forwarding registers save $(k-1)k$ pixels at a time, replacing k pixels each cycle with vertically-sorted pixels. After k inputs are diagonally sorted, the median pixel is produced.

In order to avoid data dependency issues when utilizing the sliding window, the results of each column during vertical sorting are stored into registers for future cycles. These registers are used for internal forwarding for the next $k-1$ cycles, to create the sliding window. Therefore, with the use of internal forwarding, pipelining of the filter and the data layout of the image, a total of k pixels is sent to the median filter while resulting in one pixel each cycle as throughput.

We have chosen this bitonic sorting-based method as the platform to determine the energy demands drawn from the different components of the median filter. Our experiments were done using a 3×3 window size, meaning that nine pixels must be sorted in order to find the median. However, due to our exploitation of the nature of the sliding window, we reduce the overall number of comparison units, lowering the hardware complexity. Our sorting-based network is pipelined for higher throughput, using a combination of comparators and multiplexers for the median filter and block RAM for image storage. By pipelining our architecture, we have developed a highly optimized structure which delivers 1 pixel per cycle throughput along with lower energy consumption due to our memory activation scheduling technique.

B. Power Dissipation Analysis

To determine the bottleneck on energy efficiency for median filtering and where optimizations should be done, we first perform a high-level energy analysis. Energy analysis can be equated to power analysis when finding energy hot spots. We look into the power analysis with the assumption that we are storing the image fully into memory using block RAMs. The baseline implementation is the same median filter architecture, without any memory management except for pixel placement into memory. All memory will be active at all times. There are three types of power which must be considered: computational power, interconnection power, and memory power. For an

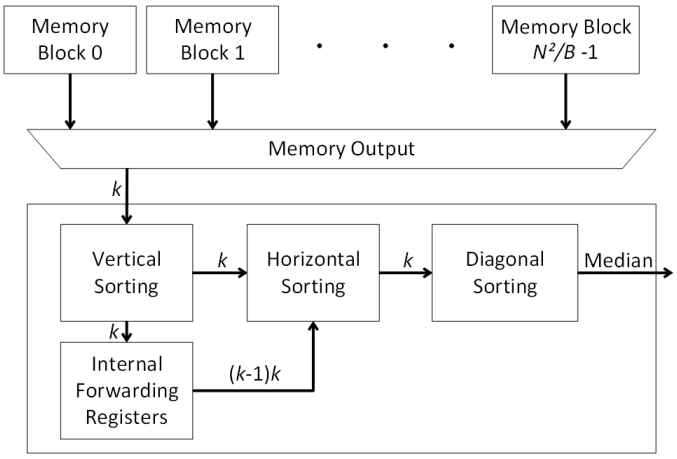


Fig. 6: Median filtering architecture

$N \times N$ image using a $k \times k$ window, the number of operations which will be performed is $N^2k^2\lceil\log^2 k\rceil$, the amount of time during filtering and storage is N^2 cycles, and the amount of memory is N^2/B, assuming that each block RAM is size B. For this high-level analysis, we assume the number of comparison units is $k^2\lceil\log^2 k\rceil$. Therefore, the power consumed by the computational units is $O(N^2k^2\lceil\log^2 k\rceil)$, assuming a unit of power to be consumed for performing one comparison operation. The power consumed by memory is $O(N^4)$, with the assumption that a unit power is consumed in a cycle for a unit of storage. We observe that communication is mainly between adjacent processing elements or with the components within the PE. The power consumed for communication is $O(N^2k^2\lceil\log^2 k\rceil)$. Given that N is much larger than k, the power consumed by the memory units is the dominant component of the overall power dissipation, in the case of storing the image into memory.

We propose memory scheduling as a technique to reduce the amount of power dissipated in the memory. The common usage of block RAMs enables all utilized memory during the filtering and storing, regardless of which blocks are actually required, wasting power keeping unneeded blocks fully active. By disengaging block RAMs that are unnecessary for the current computations or for writing new image pixels into memory, we lower the power requirements and increase the performance of memory. If memory power can be adjusted to be of a lower magnitude, then it would no longer be such a limiting factor on the scalability of the median filter design. This method lowers our required active memory to $k+1$ block RAMs for writing incoming data into memory while reading k pixels into the median filter each cycle. The resulting power consumed by memory becomes $O(kBN^2)$.

C. Memory Scheduling

To store the image into block RAM directly, we parallelize the storing of image pixels and the filtering of windows from memory. While incoming elements from the image are being stored into an active memory block, previously-stored data is being accessed for computations, outputting the resulting median after the window has gone through the entire sorting

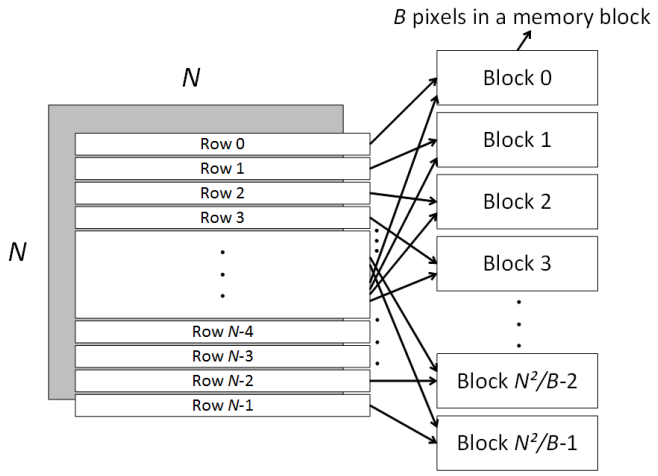


Fig. 7: Data layout

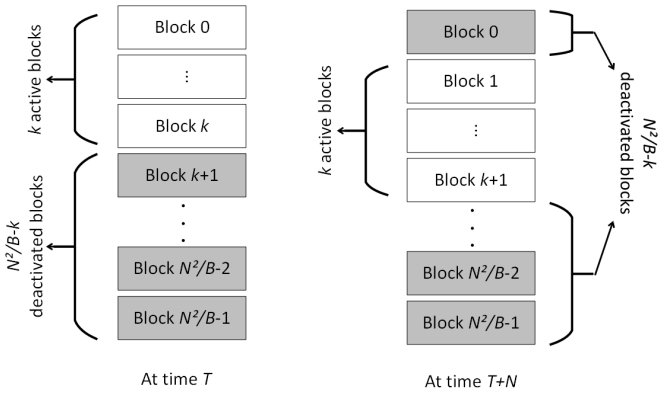


Fig. 8: Memory activation scheduling

pipeline. By parallelizing the storing and computing, we improve throughput and avoid needless inactivity of the sorting pipeline while storing new data. The other blocks of memory not being used for storing new elements or being accessed for reading in that cycle are turned off to save energy. This is done through memory activation scheduling, turning memory on and off when required, so that the minimum number of block RAM are in use at a time.

Let B be the number of image pixels that can be stored in a unit of memory (a memory block). Assuming that the data is laid out into memory as shown in Figure 7, there are a total of N^2/B memory blocks. Using our memory activation schedule, k blocks are activated each cycle for a $k \times k$ window to send k pixels in parallel per cycle to the median filter, while all other blocks are deactivated. Starting with the left-hand corner of the image, our memory activation schedule activates k memory blocks each cycle in a row-by-row fashion for a column of the sliding window's pixels as shown in Figure 8. At time T, k blocks are active for the current column of the input pixel's sliding window, and those blocks will remain active until time $T+N$, after each pixel in the current row has been filtered. Then the memory activation scheduler will deactivate the currently unused memory and activate the next block to create the new window. While storing image pixels into memory and filtering simultaneously, an additional memory block is activated for storage. The memory activation scheduler ensures that there are no conflicts between storing data and reading data during those cycles.

IV. Experiments and Analysis

Our experiments were conducted on a Xilinx Virtex 7 XC7VX980 FPGA [16] with a -2L speed grade. The experiments were done using Xilinx ISE 14.1 development tools. To look at a wide variety of potential image sizes when considering the energy implications of different memory choices, we used image sizes ranging from 128×128 pixels to 1024×1024 pixels. The median filter utilizes a 3×3 window size for all experiments, and we also adopt a wide range of standard pixel bit depths, including 8 bpp, 16 bpp, 24 bpp, and 32 bpp. The Virtex-7 device has 153K logic slices with a maximum of 54 Mbit block RAM and 880 Input/Output (I/O) pins.

While looking at power consumption, we only consider dynamic power in our experiments. Using post-place and route results, we use 20% toggle rates in all our experiments, due to it being the worst case rate for logic-intensive modules [15].

A. Peak Performance

Energy efficiency is defined as the number of operations per unit energy consumed by the design, where energy consumed by the design is (time taken by the design) $\times$ (average power dissipation of the design). Alternatively, energy efficiency of the design is power efficiency, the number of operations per second per Watt. To quantify the efficiency of our implementations, we estimated the sustained energy efficiency (percentage of the peak energy efficiency that can be sustained by the design). Peak energy efficiency is the upper bound of the performance of the platform implementing the given kernel. The peak performance depends on the target device and the IP cores used. We measured the peak energy efficiency using a minimal architecture for the processing element under ideal conditions. This minimal architecture performs only the basic operations in any kernel design. Thus, we ignore other overheads such as memory energy, I/O, access to memory, cache and other buffers that may be incurred by an implementation.

The common element to all median filter implementations is the comparison logic needed to determine the median and the minimal architecture of any median filter will consist of a comparator, along with basic input and output registers for each unit. This minimal architecture is the upper bound to any median filtering approach, and we cannot go beyond its performance on this FPGA device. The best that can be done is to approach the high energy efficiency results from this architecture. Based on experiments, along with tests with timing constraints to configure the minimal unit to achieve the highest performance, the peak performance results are 239.9 GOPS/s/W, 119.3 GOPS/s/W, 78.9 GOPS/s/W, and 51.0 GOPS/s/W for 8 bpp, 16 bpp, 24 bpp, and 32 bpp, respectively. Using these results, we can evaluate the energy efficiency of our approach in relation to this upper bound.

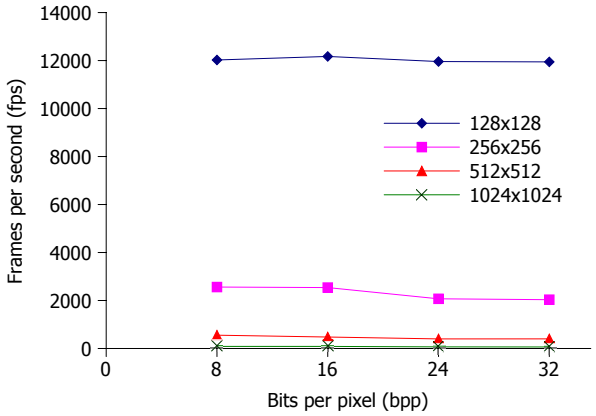


Fig. 9: Throughput of our optimized approach

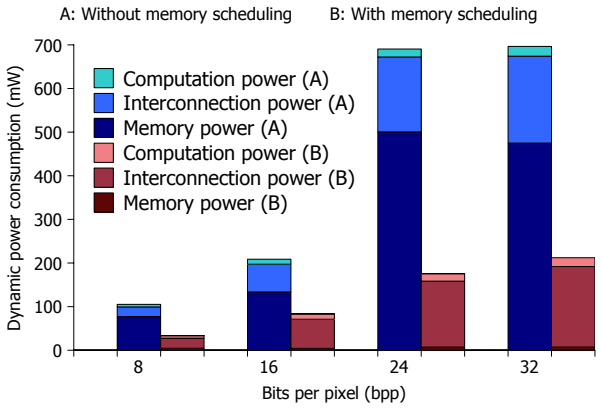


Fig. 10: Power profile of the architecture for a 512 x 512 image

In order to identify the frequency at which peak energy performance is reached, we measured the power consumed by the design while adjusting the frequency at intervals of 50 MHz, from 50 to 700 MHz. There is a small change in the architecture when adjusting the timing constraints as well, with tight constraints resulting in higher power requirements, while moderate or generous constraints can lead to a slightly larger design with regards to area and routing but less power demand. We adjusted the constraint at the same intervals as the frequency. The peak energy efficiency was observed at 650 MHz, 700 MHz, 650 MHz, and 600 MHz for 8 bpp, 16 bpp, 24 bpp, and 32 bpp, respectively.

B. Throughput

We define throughput as the frame rate or the number of frames per second. The industry standard for motion picture is 24 frames per second, and to ensure a high frequency at which imaging devices produce consecutive images, we require a minimum of 25+ frames per second from our median filter design. Our results using our memory activation scheduling technique are shown in Figure 9.

The throughput lowers as the image size increases, due to the higher demands on the resources and memory placed by larger images, as well as the increase in wiring that arises with the additional memory requirements. In the case of small images, the number of cycles required to complete the filtering of a frame is low, resulting in a high frame rate with a minimum of 11320 fps due to pipelining for a 128×128 image and a minimum of 2033 fps for a 256×256 image. Larger images require substantially more cycles for each frame, with a 512×512 image allowing for at least 400 fps, and a 1024×1024 image allowing a minimum of 63 fps.These results guarantee that our architecture can maintain at least these minimum frame rates without issue. Using our median filtering approach and memory activation scheduler, we achieve frame rates beyond 25+ frames per second for all image sizes implemented.

C. Energy Efficiency and Power Evaluation

The XPower Analyzer tool is used to measure the power consumption of our device for our experiments, and our results are presented in Figures 11, 12 13, and 14, with the metric of Giga-operations per second per Watt (GOPS/s/W) as well as a power profiling comparison in Figure 10. When calculating the energy efficiency, we utilize the power consumed by logic, signals, and block RAM. We use the image sizes of 128×128, 256×256, 512×512, and 1024×1024, along with the previously mentioned range of bpp.

We use single-port block RAMs to lower the energy cost of memory, since the reading of pixels does not coincide with the writing of incoming data. Instead, the two actions are done in different blocks of memory, allowing parallel processing of data. Memory scheduling also lowers the effect of memory on performance, however, this improvement results in the next dominating factor on energy efficiency to be communication energy, between the block RAMs and median filter. As the number of block RAMs increase, the memory power remains relatively constant, but the communication costs are more substantial due to the increasing wire lengths. We also minimize the number of block RAMs in order to offset this new performance bottleneck, improving our performance further.

Figure 10 shows the power profiles of the baseline and optimized architectures for a 512×512 image. In the baseline architecture, memory power from the fully-enabled block RAMs dominate, indicating a need for memory power reduction. By using our memory scheduling technique, we reduce the memory power significantly by an average of $45\times$, while maintaining similar computation and routing power between the two.

With the use of memory scheduling, there is a notable improvement in energy efficiency over the baseline architecture. However, with this increase in improvement over the baseline, there is also an overall decrease in energy efficiency as the image size increases. This decrease is a result of higher routing demands and the usage of large, parameterized components (such as the multiplexer for selecting pixels for the filter) implemented for the scheduler as the required number of block RAMs increases. BRAMs, by nature, are situated on the board in permanent, inflexible locations, limiting possible opportunities for reduction of wire lengths. To offset this

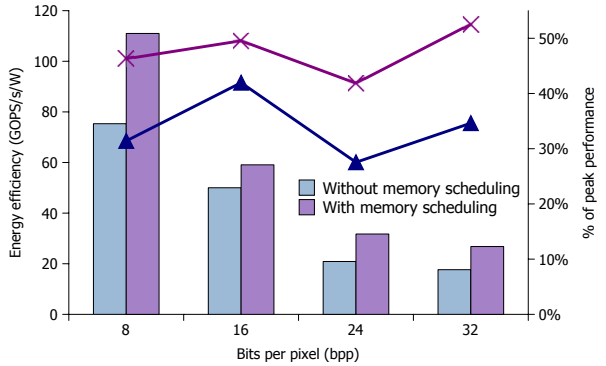


Fig. 11: Performance of a 128 x 128 image

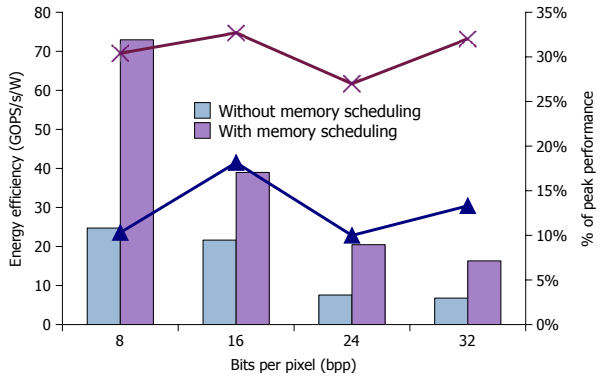


Fig. 12: Performance of a 256 x 256 image

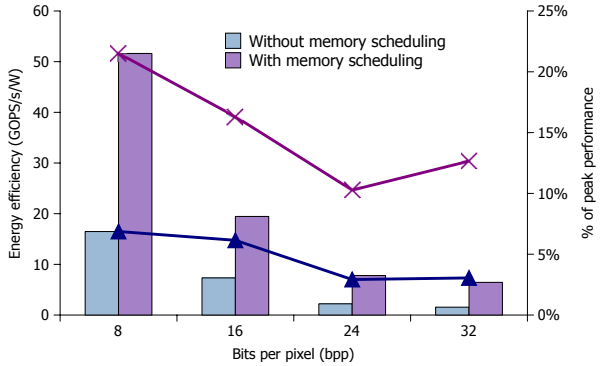


Fig. 13: Performance of a 512 x 512 image

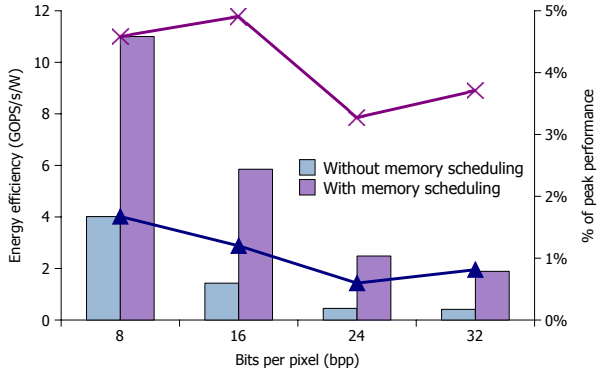


Fig. 14: Performance of a 1024 x 1024 image

disadvantage, we reduce the number of block RAMs used during our data mapping, though for large image sizes such as 1024×1024, minimization becomes a moot point due to the physical constraints on block RAM sizing. The energy efficiency is also significantly higher when there is a lower rate of bits per pixel, mainly due to the allowance of more data placed into each block of memory, lowering the routing and signal costs and decreasing the number of required block RAMs to fully store the image.

The required memory size is primarily dependent on the image size, with increasing rates of energy efficiency improvement over the baseline as the image size increases. The average improvement for a 128×128 image is $1.4\times$, for a 256×256 image is $2.5\times$, for a 512×512 is $4.0\times$, and for a 1024×1024 image is $4.2\times$. However, even though the improvement over the baseline increases with larger images, the actual amount of energy efficiency decreases significantly, limiting the real gains we can achieve when increasing the image size. Storing the image on-chip should only be done for smaller images when energy efficiency is the main concern.

The percentage of peak performance achieved is shown as two lines of matching colors with the legend, with a blue line with triangles representing the baseline architecture and a purple line with $\times$'s showing the percentage of performance achievement for the optimized architecture. Due to the high memory demands, peak performance cannot be reached, but up to 53% can be achieved using our memory scheduling technique, allowing for significantly higher energy efficiency results.

There is a dip for all image sizes when utilizing a depth of 24 bpp, since 24 is not a natural width for block RAM. Unlike the bits per pixel depths of 8, 16, and 32, there will be some waste when using a width of 24 for the memory, which results in a lower percentage of peak performance.

V. Conclusion

We developed an energy-efficient median filter architecture when storing the image on-chip, first by determining the bottleneck on energy efficiency, which was deduced to be memory power, and then by reducing the memory's power consumption using a memory activation scheduling technique. We also estimated the upper bound possible for any median filtering algorithm using the target platform, which can be used a measure of how energy efficient a technique is. We achieved up to 53% of the peak energy efficiency. We achieved a minimum of 63 frames per second (fps) for the worst case of a 1024×1024 image, which is still above the required $25+$ fps.

Acknowledgment

The authors would like to thank Yusong Hu for his comments on an earlier draft of this paper.

References

[1] Gavin L Bates and Saeid Nooshabadi. Fpga implementation of a median filter. In *TENCON'97. IEEE Region 10 Annual Conference. Speech and Image Technologies for Computing and Telecommunications., Proceedings of IEEE*, volume 2, pages 437–440. IEEE, 1997.

[2] DRK Brownrigg. The weighted median filter. *Communications of the ACM*, 27(8):807–818, 1984.

[3] Chaitali Chakrabarti. Sorting network based architectures for median filters. *Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on*, 40(11):723–727, 1993.

[4] Suhaib A Fahmy, Peter YK Cheung, and Wayne Luk. Novel fpga-based implementation of median and weighted median filters for image processing. In *Field Programmable Logic and Applications, 2005. International Conference on*, pages 142–147. IEEE, 2005.

[5] Suhaib A Fahmy, Peter YK Cheung, and Wayne Luk. High-throughput one-dimensional median and weighted median filters on fpga. *IET computers & digital techniques*, 3(4):384–394, 2009.

[6] Pekka Heinonen and Yrjo Neuvo. Fir-median hybrid filters. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 35(6):832–838, 1987.

[7] T Huang, G Yang, and G Tang. A fast two-dimensional median filtering algorithm. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 27(1):13–18, 1979.

[8] Mustafa Karaman, Levent Onural, and ABDULLAH Atalar. Design and implementation of a general-purpose median filter unit in cmos vlsi. *Solid-State Circuits, IEEE Journal of*, 25(2):505–513, 1990.

[9] W James MacLean. An evaluation of the suitability of fpgas for embedded vision systems. In *Computer Vision and Pattern Recognition-Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on*, pages 131–131. IEEE, 2005.

[10] Kemal Oflazer. Design and implementation of a single-chip 1-d median filter. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 31(5):1164–1168, 1983.

[11] Simon Perreault and Patrick Hébert. Median filtering in constant time. *Image Processing, IEEE Transactions on*, 16(9):2389–2394, 2007.

[12] Dragana Prokin and Milan Prokin. Low hardware complexity pipelined rank filter. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, 57(6):446–450, 2010.

[13] John W Tukey. Exploratory data analysis. *Reading, Ma*, 231, 1977.

[14] Zdenek Vasicek and Lukas Sekanina. Novel hardware implementation of adaptive median filters. In *Design and Diagnostics of Electronic Circuits and Systems, 2008. DDECS 2008. 11th IEEE Workshop on*, pages 1–6. IEEE, 2008.

[15] Xilinx. Power tools tutorial. http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ug733.pdf.

[16] Xilinx. Virtex-7 fpga family. http://www.xilinx.com/products/silicon-devices/fpga/virtex-7/index.htm.