A Real-time Energy-Efficient Superpixel Hardware Accelerator for Mobile Computer Vision Applications

Injoon Hong
KAIST
injoon.hong@kaist.ac.kr

Jason Clemons
NVIDIA
jclemons@nvidia.com

Rangharajan Venkatesan
NVIDIA
rangharajanv@nvidia.com

Iuri Frosio
NVIDIA
ifrosio@nvidia.com

Brucek Khailany
NVIDIA
bkhailany@nvidia.com

Stephen W. Keckler
NVIDIA
skeckler@nvidia.com

ABSTRACT

Superpixel generation is a common preprocessing step in vision processing aimed at dividing an image into non-overlapping regions. Simple Linear Iterative Clustering (SLIC) is a commonly used superpixel algorithm that offers a good balance between performance and accuracy. However, the algorithm's high computational and memory bandwidth requirements result in performance and energy efficiency that do not meet the requirements of real-time embedded applications. In this work, we explore the design of an energy-efficient superpixel accelerator for real-time computer vision applications. We propose a novel algorithm, Subsampled SLIC (S-SLIC), that uses pixel subsampling to reduce the memory bandwidth by 1.8×. We integrate S-SLIC into an energy-efficient superpixel accelerator and perform an in-depth design space exploration to optimize the design. We completed a detailed design in a 16nm FinFET technology using commercially-available EDA tools for high-level synthesis to map the design automatically from a C-based representation to a gate-level implementation. The proposed S-SLIC accelerator achieves real-time performance (30 frames per second) with 250× better energy efficiency than an optimized SLIC software implementation running on a mobile GPU.

1. INTRODUCTION

The recent improvements in mobile processing capability has allowed computer vision to be deployed in mobile and embedded devices for applications such as autonomous vehicles, augmented reality, and mobile robotics. The computational requirements for such vision applications is growing rapidly due to the increased resolution of camera sensors, such as 1920×1080 full HD imaging.

Many computer vision applications use a class of segmentation algorithms known as Superpixel (SP) segmentation. SP segmentation groups the pixels of an image into atomic groups based on spatial locality and characteristics such as color. These regions can be used to reduce the complexity of image processing tasks later in the computer vision pipeline. Consequently, SP segmentation has become a key component in many computer vision algorithms such as object classification, depth estimation, and region segmentation.

Simple Linear Iterative Clustering (SLIC) has become one of the most commonly used SP algorithms [1]. SLIC provides a good balance between accuracy and computational complexity for many applications. However, the recent increases in camera resolutions make it difficult for SP algorithms, including SLIC, to achieve real-time performance (30 frames per second or fps). While SLIC segmentation can significantly reduce the complexity of downstream processing, the SLIC algorithm itself operates directly on raw data. SLIC takes 5500ms to process a 1920×1080 image on an Intel i7-4600M CPU, 1600ms on the mobile GPU of an NVIDIA Tegra K1, and 22.6ms on the high-powered NVIDIA Tesla K20. While the big discrete GPU (K20) can meet the real-time requires of 33ms (30 fps), its 80W power consumption far exceeds the budget for mobile computing platforms.

To achieve real-time mobile performance for SLIC-based SP segmentation, we have developed both a new algorithm named subsampled SLIC (S-SLIC) and an S-SLIC hardware accelerator for mobile systems. We use subsampling to reduce the computational and memory bandwidth requirements of SLIC. Our accelerator provides efficient computation of superpixels at 30 fps while providing 250× better energy efficiency than a SLIC implementation on a mobile GPU. The key contributions of this work are:

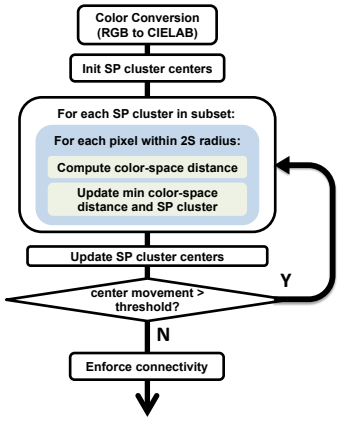
- We propose an efficient SLIC algorithm named subsampled SLIC (S-SLIC) and explore different subsampling mechanisms to reduce the computational and memory bandwidth requirements.
- We design, perform an in-depth design space exploration, and analyze design trade-offs for a S-SLIC accelerator.
- We present an evaluation of our design in a 16nm FinFET technology using a systematic framework consisting of (i) the Mentor Catapult high-level synthesis tool to automatically generate RTL from a synthesizable C-based implementation, (ii) Synopsys Design Compiler for logic synthesis, and (iii) Synopsys Primetime-PX to perform power analysis.

2. SLIC SUPERPIXELS

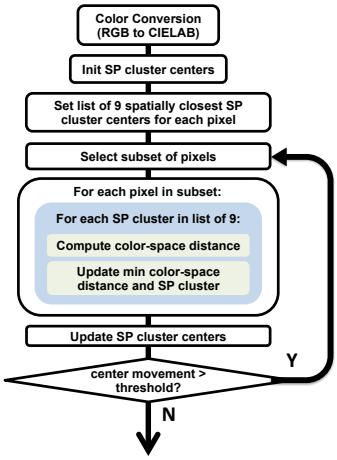
SLIC is an adaptation of k-means for fast superpixel (SP) generation [1]. Figure 1a shows the algorithm's flowchart. The user specifies the number of superpixels K the algorithm is to produce. The algorithm first converts an image composed of N pixels to the CIELab color space. Starting from $[r,g,b]$ in the $[0,1]$ range, this is

DAC '16, June 05-09, 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2897937.2897974



(a) Basic SLIC (Center Perspective Architecture (CPA))



(b) S-SLIC with Pixel Perspective Architecture (PPA).

Figure 1: Flowcharts of SLIC algorithms.

achieved by computing:

$$x' = \begin{cases} x/12.92 & \text{if} \quad x \le 0.04045 \\ \left[(x+0.05)/1.055\right]^{2.4} & \text{if} \quad x > 0.04045 \end{cases} \qquad x = \{r,g,b\} \quad (1)$$

$$\begin{bmatrix} X & Y & Z \end{bmatrix}^T = \mathbf{M} \cdot \begin{bmatrix} r' & g' & b' \end{bmatrix}^T \quad (2)$$

$$L = 116 \cdot f_Y - 16, \quad a = 500 \cdot (f_X - f_Y), \quad b = 200 \cdot (f_Y - f_X) \quad (3)$$

$$f_W = \begin{cases} \left(\frac{W}{W_r}\right)^{1/3} & if \quad \frac{W}{W_r} > 0.008856 \\ \frac{903.3 \cdot (W/W_r) - 16}{116} & otherwise \end{cases}, \quad W = \{X,Y,Z\}, \quad (4)$$

where $\mathbf{M}$ is a 3×3 matrix and $[X_r, Y_r, Z_r]$ is the reference white. The SP centers are initialized on a regular grid, with a spacing of $S = \sqrt{N/K}$ pixels. The i-th SP is identified by $[L_i, a_i, b_i, x_i, y_i]^T$, where the first three coordinates represent the average color of the pixels of the SP and the last two coordinates its center. Each SP center is then moved to the local minimum of the gradient image in a 3×3 neighborhood, to avoid initialization on an edge or a noisy pixel.

At each iteration, SLIC loops over all the SPs. For each of them, it computes the color-space distance of each pixel within a $2S \times 2S$ rectangle centered in the SP as:

$$d = \sqrt{d_c^2 + m^2 \cdot \left(d_s^2/S\right)^2}, \quad (5)$$

where d_c and d_s are respectively the color and spatial distance of the pixel from the SP, while m balances the importance of these two factors (generally set between 1 and 40).

Each pixel is assigned to the SP with smallest distance d. Two memory buffers (as large as the image) are required to store the minimum distance and the corresponding SP for each pixel. Once all the SPs have been considered, the 5D coordinates of each SP are recomputed using the current member pixels. SLIC iterates until either the change in the SPs is below a threshold or the maximum number of iterations is reached. At convergence, a final step is performed to enforce the connectivity, ensuring that any stray pixels that may still be disjoint are assigned to the closest large SP.

3. SUBSAMPLED SLIC

Partially inspired by the stochastic gradient descent [2] and OS-EM[4] approaches, we developed S-SLIC, a new way to compute SLIC superpixels that is more efficient than the original SLIC. At each iteration, we use a subset of the image pixels for computing the SP centers, which allows us to update each SP with a fraction of the distance computations and memory bandwidth while still computing the SP location accurately.

Choosing the proper subsampling strategy is fundamental to guaranteeing the convergence of the iterative algorithm [4, 2]. We primarily consider a pixel perspective architecture (PPA), shown in Figure 1b. The image pixels are split into subsets of equal size. At each iteration, a different subset is used to update the SPs. The subsets are traversed in a round-robin fashion to guarantee that all image pixels are considered. We also examined a SP Center Perspective Architecture (CPA) in which the SPs are split into subsets of equal size. At each iteration, a different subset is used to update the SPs. The SP subsets are traversed in round-robin order. As discussed in Section 4, the two methods, the PPA and the CPA, differ in the amount of data required from DRAM and the amount of computation. We focus primarily on the pixel perspective approach as our results show that it is more efficient.

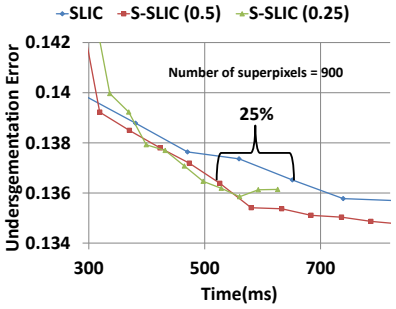
By adopting the proper subsampling strategy, S-SLIC leads to improved performance in two key metrics, as shown in Figure 2. The first is the undersegmentation error (USE) [1] which considers the superposition of the computed and ground truth SPs. USE is higher when the location and size of the computed SPs differ significantly from the ground truth. Figure 2a shows USE for SLIC and two variants of the pixel perspective S-SLIC, when dividing the image pixels into two (S-SLIC (0.5)) or four (S-SLIC (0.25)) equally sized subsets. The figure shows that S-SLIC achieves the same USE of SLIC in a 25% shorter time.

The second metric is the boundary recall [1], which is a measure of how well the SP boundaries align with the ground truth. The higher the boundary recall, the more the SP boundaries match the ground truth. Figure 2b shows that for the same boundary recall, S-SLIC (0.5) has a 15% shorter execution time than SLIC. Thus, S-SLIC provides the same error rates for a lower execution time.

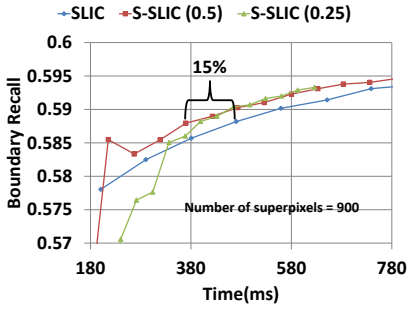
4. SLIC ACCELERATOR

4.1 Analysis of S-SLIC Algorithm

Table 1 shows the time breakdown of the proposed PPA of S-SLIC measured on an Intel Core i7-4600M CPU using the Berke-



(a) Undersegmentation error versus runtime.



(b) Border recall versus runtime.

Figure 2: CPU Performance of SLIC and pixel perspective S-SLIC for subsampling ratios 0.5 and 0.25. The test corpus includes 100 images from the Berkeley segmentation dataset with $K = 900$ superpixels.

Table 1: Time breakdown of SLIC and S-SLIC implementations.

	Color Conversion	Distance + Min	Center Update	Other
SLIC	23.4%	65.9%	10.2%	0.5%
S-SLIC	18.7%	59.7%	17.9%	3.7%

ley benchmark [6]. As in Table 1, the most dominant process in S-SLIC is the computation of the color-space distance and the assignment to an SP which takes 59.7% of total processing time. The second dominant phase is the center updating process (17.9%) which becomes a much more time consuming process in S-SLIC compared to the original SLIC (10.2%) since S-SLIC updates the centers more frequently than the original SLIC with the same number of full iterations. The third dominant phase is color conversion which takes 18.7% of total processing time. The remaining execution (3.7%) includes the connectivity enforcement, and some initialization tasks shown in Figure 1, are not covered by the proposed accelerator in Section 4.3

4.2 Center and Pixel Perspective Architecture

Figure 3 shows an overview of the CPA and the PPA. The CPA in Figure 3 first reads a 2Sx2S 2D patch surrounding a given SP center point. Next, it calculates the color-space distances between the center and the pixels in the 2D patch. After the distance computation over all the SP center points, each pixel is assigned to the SP cluster whose center is the smallest distance away. In contrast, the PPA reads the 9 nearest centers for a given pixel to calculate the distances, where the the 9 centers can be found by using the closest SP centers at the start of processing. 9 is the minimum number of nearest centers that can be considered to cover all possible pairs of center and pixel in the original CPA SLIC. Table 2 summarizes computational complexity and required memory traffic of the two approaches. Since the CPA needs large overlapped image area, 2SxS, between two adjacent 2Sx2S patches, the input image data must be read multiple times per iteration which results in higher memory bandwidth (more than 300MB/iteration in 1080p). However, the PPA reads the image once per iteration, needing 3x lower memory bandwidth than CPA, 100MB per iteration in 1080p. The reduction in memory bandwidth comes at the expense of more computational intensity. The PPA needs 2.25× more operations for computing distances than the CPA, because the PPA needs nine color-space distance calculations for a given pixel, compared to four distance calculations required on average by the CPA. The PPA shows almost same but slightly better SLIC accuracy than the CPA since the PPA considers more distance values in SP decision. A simple energy model provides insight into which architectural approach would minimize energy consumption. We

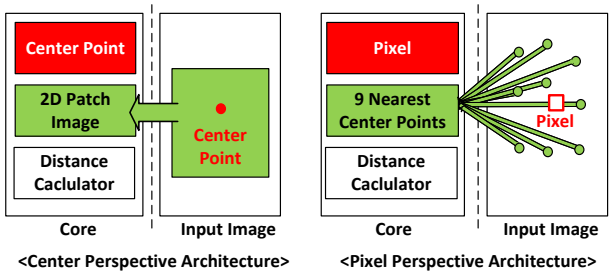


Figure 3: Pixel (PPA) and center (CPA) perspective architectures.

Table 2: Analysis of CPA and PPA implementations.

	CPA	PPA
Memory Bandwidth	318 MB/iteration	100 MB/iteration
Operation count	58M OPs/iteration	130M OPs/iteration

assume the average energy per arithmetic operation in the SLIC algorithm is similar to the energy of an 8b integer add and that the energy of an 8b DRAM reference is 2500x larger the energy of an 8b add [3]. Based on this simple model, we expect total energy per iteration for the two approaches in Table 1 to be dominated by the energy of accessing data from DRAM; thus we adopt the lower-bandwidth PPA approach for the SLIC accelerator design shown in Figure 1b.

4.3 Overall Hardware Architecture

The proposed hardware accelerator architecture shown in Figure 4 consists of 5 components: a Finite State Machine (FSM) host controller, a fixed-point color conversion unit with look-up tables (LUTs) for efficient power-function implementation, four scratch pad memories (three for storing color channels and one for storing SP index results), a Cluster Update Unit to accelerate the minimum and color-distance operations, and a Center Update Unit to find new center points for each iteration.

The first step in the S-SLIC algorithm is color conversion. Input RGB image data is loaded from external memory in raster-scan order where single-byte RGB values per pixel are stored contiguously. R, G, and B input colors are loaded into the channel memories 1, 2, and 3, respectively. Once pixel values are loaded, the color conversion unit starts reading from the scratch pads to fetch each pixel's RGB values. The color conversion unit converts from RGB to CIELAB color space using LUTs, storing output values L, a, and b back into channel memories 1, 2, and 3, respectively. After color conversion is run on all pixels in the image, the FSM moves the S-SLIC accelerator to the next step.

Next, the accelerator performs the initial assignment of the 9 closest SP centers for a given pixel. Although SLIC executes this

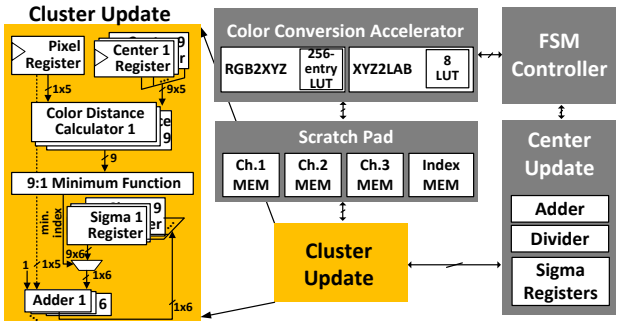


Figure 4: S-SLIC accelerator architecture.

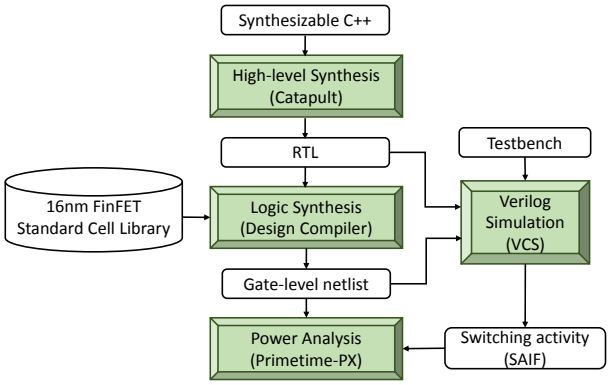


Figure 5: S-SLIC accelerator Evaluation Methodology.

step with each image, our S-SLIC implementation precomputes these values. We found that statically assigning these values has minimal effect on the accuracy of the algorithm. The image is statically split into tiled regions based on the initial 9 closest SPs. These regions are computed offline and stored in external memory. External memory also holds a copy of the pixel SP assignments initialized to zero.

The next step is to update SPs. First, tile regions are loaded into scratch pad memories: the L, a, and b data values are written into channel memories 1, 2, and 3, respectively, while the SP indices are written into the index memory. The color and spatial information for the 9 initial closest SPs are loaded into the center registers from external memory based on the tile being processed. The current accumulations for the 9 SPs in the cluster update unit are loaded from the center update unit. Once loaded, the FSM instructs the cluster update unit to begin processing.

The cluster update unit starts by loading pixel data into pixel registers: color values are loaded from the scratch pads while the pixel location itself is supplied by the FSM. Five 8-bit registers are needed for the current pixel's data and 45 (5×9) registers for the cluster center data. While a tile is being processed only the five 8-bit registers are updated for each pixel. The color distance unit then computes the color distance and spatial distance of the pixel from the center using Equation 5 based on these registers. Each unit accepts five registers per pixel and five registers for a single center and returns the 8-bit distance. Once all nine color distance calculations are computed, they are passed to the 9:1 minimum function unit, returning the SP center index associated with the smallest distance. This index is used to select the sigma register for accumulating the pixel values for a SP. Each sigma register holds six fields: the accumulated L, a, and b color information, the accumulated x,y location information, and the number of pixels assigned to the associated SP. Once the correct sigma register is selected, each field is updated with the new pixel information, requiring six additions. The SP index for the pixel is also written back to the index memory scratch pad.

Once all pixels in a tile have been processed by the cluster update unit, SP accumulations are stored back to the center update unit and the index memory scratchpad is stored back to external memory. The cluster update unit then moves to the next tile. After the cluster update unit processes the entire image, the center update unit begins computing the new centers for each SP. The center update unit iterates over its sigma registers to compute the average for each SP. The resulting new centers are written to the external memory for use in the next iteration of the cluster update.

The system continues in this manner until the desired number of iterations is reached or theof difference between the old and new SP centers is below a threshold. At this point, the final assignment for each pixel will be stored in the external memory.

5. EXPERIMENTAL FRAMEWORK

The proposed S-SLIC accelerator design was prototyped with a 16nm FinFET standard-cell library using the methodology shown in Figure 5. We designed the S-SLIC accelerator in synthesizable C using hardware libraries and used the high-level synthesis tool Catapult to generate RTL. We then used Synopsys Design Compiler to perform logic synthesis, targeting 1.6GHz at 0.72V. The scratchpad memories in the proposed accelerator were realized using synchronous RAMs with separate read-write ports. We used Synopsys VCS to simulate the netlist and extract switching activity information (SAIF), which was then used to perform power analysis in Synopsys Primetime-PX assuming 0.72V operation. The design was highly parameterized to allow in-depth design space exploration of the accelerator by varying the number of cores, number of SIMD ways, memory size, and bit-widths of different operations. The HLS-based implementation, design space exploration, logic synthesis, and power analysis steps took 3-4 person weeks of design effort. In our experiments, we used 200 images from the Berkeley image segmentation dataset [6]. For baseline comparison, we ran the SLIC algorithm on two different GPUs: Tegra K1 (mobile GPU) and Tesla K20 (server class GPU). We used CPU Intel Core i7-4600M CPU to evaluate the run time versus error tradeoffs for the SLIC and S-SLIC algorithms.

6. EVALUATION

To optimize the performance of the S-SLIC accelerator, we performed a design space exploration to determine the proper bit-width for data, computational configuration of the cluster update and size of memory buffers for computation. These three key items have a direct effect on performance, energy, and area of the accelerator.

6.1 Bit Width Exploration

The number of bits used to represent the data is a critical decision for energy-efficient accelerator design. We performed an analysis of the error in the output given various data sizes and types, analyzing both undersegmentation error and boundary recall. S-SLIC error does not increase noticeably as numerical precision is reduced from 64-bit floating-point to 8-bit fixed-point. At 8-bit fixed point representation we see only 0.003 larger undersegmentation error, and only 0.001 smaller boundary recall, compared to the 64-bit double-precision S-SLIC implementation. The primary reason for the robustness of S-SLIC to bit width is that the S-SLIC accuracy is determined by the relative color-distance comparison results rather than the absolute color-distance results. Therefore, even though the 8-bit fixed-point generates large error in the distance values, it achieves small differences in the results from the distance comparison. At 7-bit precision and below, the increase

Table 3: Cluster Update Unit configurations.

Distance calculator	1 way	9 way	1 way	1 way	9 way
Minimum unit	1 way	1 way	9 way	1 way	9 way
Adder unit	1 way	1 way	1 way	6 way	6 way
Area (mm^2)	0.0020	0.0149	0.0023	0.0025	0.0156
Power (mW)	3.3	3.6	3.2	3.25	30.9
Latency (cycles)	27	19	20	22	7
Throughput (pixels/cycle)	1/9	1/9	1/9	1/9	1
Time (ms)	11.8	11.8	11.8	11.8	1.3
Energy (μJ)	38.9	42.5	37.5	38.3	40.6

in error begins to be noticeable. The selection of an 8-bit based datapath affected the color conversion hardware design. To accommodate the 8-bit width, we use a look-up table (LUT) based design in the color conversion step to achieve better energy-efficiency. We adopt a 256-entry LUT for the power function used in the 8-bit RGB to XYZ conversion, (Equation 1), and an 8 component piecewise linear LUT approximation of the power function used in the XYZ to LAB conversion, (Equation 4), presented in Section 2.

6.2 Parallelism Exploration

The Cluster Update Unit performs three primary functions: color distance calculation, minimum distance computation, and sigma accumulation updates. The implementation of each function involves design tradeoffs between area and throughput. For example, an iterative unit can take multiple cycles to process each pixel or can be parallelized to support single-pixel-per-cycle throughput. The color distance calculation can compute all 9 distance functions in parallel or could time-multiplex the same hardware to iterate over the 9 calculations. The minimum distance computation could be implemented using a tree structure with 9 inputs to maximize parallelism or could iterate over 9 cycles with a single compare ALU. Finally, for the sigma accumulations, the hardware implementation must update the color and location accumulations along with the SP size. This requires 6 additions: 3 for color, 2 for location and 1 for SP size. These additions could be computed in parallel across 6 adders or could be time-multiplexed onto a single adder over 6 subsequent cycles.

Experimental results for five Cluster Update Unit configurations are shown in Table 3. For each configuration, we report area, power, latency, throughput, and computation time and energy for processing 1 iteration of a 1920x1080 image. The configurations are named based on the ways of parallelism being exploited. For example, the 1-1-1 way implementation uses iterative hardware for each function, achieving a throughput of 9 cycles per pixel. Meanwhile, the 9-9-6 way implementation exploits parallelism for all three functions to achieve a throughput of one pixel per cycle. All configurations use the same synthesizable C++ source code. Loop unrolling directives are used to control the choice of mapping each function to either iterative time-multiplexed or parallel fully-pipelined hardware. HLS tools are used to automatically schedule and pipeline the computation assuming 1.6 GHz operation with the latency reported for each configuration.

The most efficient configuration from Table 3 is the fully-pipelined 9-9-6 way design, which optimizes for parallelism at the expense of higher area. The 9-9-6 way design is 7.8× higher area and 9.4× higher power compared to the 1-1-1 way design. However if offers 9× increase in throughput, which is critical for real-time performance. This improved performance causes the energy to increase only marginally compared to other configurations. The 9-1-1, 1-9-1, and 1-1-6 way designs have imbalanced throughput, so would not be chosen for a practical design. However, we included the results to show the area cost of pipelining each of the three functions. Due to its energy efficiency and high throughput, we chose the 9-9-6 way configuration for the Cluster Update Unit architecture.

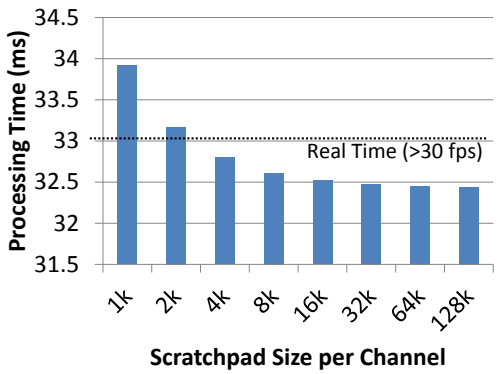


Figure 6: Performance of channel buffer sizes; 5,000 superpixels in a 1920×1080 image.

Table 4: Performance summary of best S-SLIC configurations.

Resolution	1920×1080	1280×768	640×480
Number of SPs (K)	5000	5000	5000
Memory per Channel Buffer	4kB	1kB	1kB
Number of cores	1	1	1
Area	0.066mm^2	0.053mm^2	0.053mm^2
Power	49mW	46mW	50mW
Latency	32.8ms	25.4ms	19.7ms
Throughput	30.5 fps	39.0 fps	50.3 fps
Energy	1.6mJ/frame	1.17mJ/frame	0.98mJ/frame
Perf/Area	461 fps/mm^2	747 fps/mm^2	963 fps/mm^2

6.3 Buffer Size Exploration

The final component of the design space exploration is the size of the buffers for pixel and label data. The S-SLIC accelerator requires 4 memory buffers, one for each color channel and one for the pixel labels. The buffers are used to hold the color converted pixels and the cluster updates as well. As the buffers are largely used to rate-match the data between stages of the algorithm, large buffers can reduce the rate of DRAM transfers, but require more energy and area.

Figure 6 shows the execution time per frame as the buffer size per channel is varied from 1kB to 128kB, for the 9-9-6 cluster configuration with HD resolution and 5,000 superpixels. We assumed that peak external bandwidth is 256b/cycle and memory latency is 50 cycle latency for this analysis. To achieve real-time performance, the buffer size must be at least 4kB. As larger buffers provide only slightly better frame time at the cost of larger area and energy, we choose 4kB buffers. In the case of the 4kB buffer size, memory access takes 35% of total execution time.

The overall computation and memory configuration process a system that achieves 1.6mJ/frame for real-time S-SLIC based segmentation on full HD images. The accelerator architecture can scale gracefully down to lower resolution image streams by reducing the buffer sizes and ultimately reducing the clock rate.

Table 4 shows the accelerator parameters and performance on HD, 720p, and VGA image sizes. The power and energy includes use of all the units of the S-SLIC accelerator. The power for each unit is computed using the peak active power from power analysis in Synopsys Primetime-PX and multiplying by the utilization. We assume the external memory and scratch pads are at full utilization.

7. ACCELERATOR PERFORMANCE

Table 5 compares a large GPU (NVIDIA Tesla K20), a GPU on a mobile SOC (NVIDIA TK1), and the proposed S-SLIC accelerator. The GPUs incorporate large on-chip storage structures (6,320kB in K20 and 368kB in TK1) to support the general-purpose program-

Table 5: GPU, mobile GPU, and S-SLIC accelerator performance.

	Tesla K20	TK1	This Work
Algorithm	SLIC	SLIC	S-SLIC
Resolution	1920×1080		
Number of SPs (K)	5000		
Technology	28nm (0.81V)	28nm (0.81V)	16nm (0.72V)
On-chip memory	6320kB	368kB	20kB
Core count	2496	192	1
Average power	86W	332mW	49mW
Power (normalized)	39W	150mW	50mW
Latency	22.3ms	2713ms	32.8ms
Energy/frame (normalized)	867mJ/frame	407mJ/frame	1.6mJ/frame

ming model, including register files, scratchpad memory, level-1 cache, and level-2 cache. They also include a large number of general purpose math units aimed at single and double-precision floating point (2496 CUDA cores in K20 and 192 in TK1). The S-SLIC accelerator requires only 20kB of on-chip storage to buffer pixel data flowing through the pipeline and a single "core" that processes pixels efficiently in a pipelined manner. In addition, the S-SLIC accelerator uses an 8-bit fixed-point precision data-path with optimized compound operations.

We computed the latency of the S-SLIC accelerator as the sum of the computation and memory access times for color conversion and cluster update. Our evaluation shows that color conversion is highly efficient and consumes only 1.4ms. Cluster update, on the other hand, is computational expensive and it involves performing distance calculation, minimum distance computation, and sigma and center updates over the entire image for 9 iterations along with frequent memory access. Cluster update consumes 31.4ms with memory accesses taking 11.1ms and computation consuming 20.3ms. For baseline comparisons, we executed the GPU SLIC algorithm on the K20 and TK1 hardware, measuring both performance and power consumption. While the K20 can achieve more than 30 frames per second, it requires an enormous power budget. TK1 requires much less power, but misses the real-time frame rate by a factor of 80. The normalized power row estimates the power consumption of the GPUs were they to be running in a 16nm process instead of the 28nm process. The normalization includes multiplicative factors of 1.25 for $voltage^2$ and 1.75 for *capacitance*, for a total of 2.2. As a result, the S-SLIC accelerator in its best configuration is over 500× more energy efficient than K20 and over 250× more efficient than K1, while meeting the 30 frames per second requirement. As the GPU in K20 is hundreds of mm^2 and the GPU in K1 is tens of mm^2, the estimated $0.066mm^2$ required for the S-SLIC accelerator is extremely small.

8. RELATED WORK

Because superpixels are an important and versatile method of image segmentation, numerous algorithms have been developed to generate them. Achanta et al. developed the widely used SLIC algorithm and demonstrated its merit over other superpixel algorithms [1]. While hardware accelerators have been designed for video segmentation [5, 9], our work is the first to develop a hardware accelerator for SLIC.

SLIC has been accelerated through algorithmic modifications along multiple dimensions. A parallel implementation for GPGPUs [8] called gSLIC uses the assignment of each pixel to one of the 9 closest superpixels during initialization, then adopts the implementation of the original SLIC algorithm. The pixel perspective (PPA) version of S-SLIC uses a similar superpixel assignment algorithm while also applying pixel subsampling to decrease the amount of computation and memory bandwidth.

Preemptive SLIC optimizes computation by halting the update of individual clusters when there is little to no difference in the cluster center location[7]. This approach avoids performing computation on pixels and regions that are not changing clusters or locations, respectively. The optimization of Preemptive SLIC is orthogonal to those performed by S-SLIC. While the two techniques could be combined, the analysis of this combined algorithm is beyond the scope of this work.

9. CONCLUSIONS

Superpixel (SP) generation is an important algorithm for image segmentation in computer vision applications. This paper first describes our novel augmentations to the commonly used Simple Linear Iterative Clustering (SLIC) algorithm called S-SLIC that exploits subsampling to reduce memory bandwidth and computation time without sacrificing the quality of result. We then designed an accelerator for S-SLIC and implemented it using a high-level synthesis design flow targeting a 16nm FinFET technology. Our results show that the S-SLIC accelerator can achieve real-time performance (30 frames per second) in $0.066mm^2$ and achieve energy efficiency 250× that of a mobile GPU. We expect that this accelerator can be incorporated into the architecture of SoCs aiming to accelerate computer vision applications.

10. REFERENCES

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk. SLIC Superpixels Compared to State-of-the-Art Superpixel Methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(11):2274–2282, November 2012.

[2] L. Bottou. Stochastic Gradient Tricks. In *Neural Networks, Tricks of the Trade, Reloaded*, Lecture Notes in Computer Science (LNCS 7700), pages 430–445. Springer, 2012.

[3] M. Horowitz. Computing's Energy Problem (and what we can do about it). In *International Solid-State Circuits Conference (ISSCC)*, pages 10–14, February 2014.

[4] M. Hudson and R. Larkin. Accelerated Image Reconstruction using Ordered Subsets of Projection Data. *IEEE Transactions on Medical Imaging*, 13(4):601–609, 1994.

[5] H. Jiang, H. Ardo, and V. Öwall. A Hardware Architecture for Real-Time Video Segmentation Utilizing Memory Reduction Techniques. *IEEE Transactions on Circuits and Systems for Video Technology*, 19(2):226–236, February 2009.

[6] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics. In *International Conference on Computer Vision (ICCV)*, pages 416–423, November 2011.

[7] P. Neubert and P. Protzel. Compact Watershed and Preemptive SLIC: On Improving Trade-offs of Superpixel Segmentation Algorithms. In *International Conference on Pattern Recognition (ICPR)*, pages 996–1001, August 2014.

[8] C. Y. Ren and I. Reid. gSLIC: A Real-time Implementation of SLIC Superpixel Segmentation. *Oxford University Technical Report*, 2011.

[9] I. Yasri. An FPGA Based Hardware Accelerator for Real Time Video Segmentation System. In *International Conference on Advanced Computer Science and Information System (ICACSIS)*, pages 63–68, December 2011.