Fast LBP Face Detection on low-power SIMD architectures

Olexa Bilaniuk, Ehsan Fazl-Ersi, Robert Laganière
University of Ottawa
Ottawa, ON, Canada
obila060|laganier@uottawa.ca

Christina Xu, Daniel Laroche, Craig Moulder
CogniVue Corporation
Gatineau, QC, Canada
cxu|dlaroche|cmoulder@cognivue.com

***Abstract*—This paper presents an embedded implementation of a face detection method based on boosted LBP features for Single Instruction Multiple Data (SIMD) architectures. The implementation exploits parallelism and data reuse in the detection algorithm and is integrated into CogniVue's Gen-1 APEX platform, which uses a SIMD design and is extremely energy efficient. The proposed embedded face detection system runs at 5 VGA frames per second, while providing similar accuracy to the PC version of the LBP face detection algorithm included in the OpenCV library.**

***Keywords*-face detection; LBP features; SIMD architecture;**

I. Introduction

One of the most fundamental requirements for many computer vision systems that deal with humans is face detection, that is, the capability of localizing human faces in an image or a video stream. In the last decade, face detection has received significant attention in academia and industry, mainly due to its wide range of applications, from surveillance and biometrics to human-computer interaction and digital photography.

The research on face detection, for the most part, has been focused on designing new algorithms or improving the accuracy of the existing methods. Therefore, the majority of the available face detection methods are software solutions designed for general purpose computational processors that are expensive, power demanding, and difficult to integrate with other devices (e.g., IPTVs and Set-top boxes) and technologies (e.g., Smart IP Cameras). Given that face detection is often being used as a primary module in higher level systems (e.g., face recognition, face verification, face tracking, etc.), it is crucial to have embedded solutions specifically optimized to detect faces as fast as possible to leave more time and resources to the remaining modules involved in the systems (e.g., face recognition or tracking). While an embedded face detection solution lowers the hardware and energy costs significantly, as it requires only a subset of hardware components (in comparison to general purpose computer based solutions), it facilitates the integration with other technologies, such as security cameras, TVs, digital cameras, etc., to create smart devices.

In this paper, we present an embedded face detection system based on the widely popular face detection method of Viola and Jones [1], where a set of simple features are computed and combined to form a strong classifier for fast and accurate face detection. While in the original method, Viola and Jones suggested the use of Haar-like features, in our implementation we focus on a variant of the algorithm, which uses features based on the Local Binary Pattern (LBP) operator [2]. LBP features are very appropriate for embedded systems in that they are both local and fast-to-compute. Furthermore, since LBP features are more complex and informative than Haar-like features, often a small number of them are sufficient to produce a strong classifier for face detection. As a result, while the classifiers based on LBP features perform almost as accurately as classifiers based on Haar-like features, they are often about 10 times more compact (i.e., with lower number of features) and usually perform much faster on most hardware platforms (including ordinary PCs).

Our main contribution in this paper is to propose an embedded implementation for Single Instruction Multiple Data (SIMD) architectures that exploits parallelism and data reuse in this face detection algorithm. The proposed implementation runs at 5 VGA frames per second on CogniVue's CV2201 APEX core, which uses less than 0.5% of the power that an ordinary CPU uses, and around 6% and 1.5% of the power that a design based on FPGA [3] or GPU [4] uses, respectively. The CV2201 APEX design is a first generation version of massively parallel processor by CogniVue and significant improvements have occurred since this novel architecture was introduced many years ago. Gen-1 APEX was initially used primarily for multi-standard video encoding/decoding applications and then in recent years more focused on accelerating image and vision processing based applications.

The remainder of this paper is organized as follows. In the next section, we review the state-of-the-art embedded face detection systems. Section 3 briefly describes the Viola and Jones face detection algorithm based on LBP features. Our proposed implementation is presented in Section 4, followed by experiments and results in Section 5. Finally, Section 6 concludes this paper by discussing the method and outlining some of the potential directions for future work.

II. Background

The majority of the existing face detection methods use the sliding window technique, where the image is scanned with a fixed-size rectangular window. A classifier is applied to the sub-image defined by the window, and returns the probability that the window bounds a face. The process can be repeated on successively scaled and/or rotated copies of the image so that faces can be detected at any size and/or orientation. Various methods of this type often differ on their choice of the classifier. While some approaches use feature-based techniques, approaches based on neural-networks (e.g., [5]) and statistical-learning-based techniques (e.g., [6]) have proven to be more effective.

In a widely influential approach [1], Viola and Jones suggested boosted cascade of weak classifiers for face detection, where each weak classifier uses a set of simple and fast-to-compute features to detect almost all faces while rejecting a certain portion of non-face image regions. This method, with some variations [7], has been integrated into the Open Source Computer Vision Library (OpenCV). The speed and accuracy of this method has prompted many researchers in the field of embedded systems to work on various implementations that facilitate the porting of this algorithm to embedded systems.

In [8], Theocharides et al. proposed an Application-Specific Integrated Circuit (ASIC) architecture which parallelizes the access to image data to facilitate the porting of the face detection algorithm. They reported a computational speed of 52 FPS, but they did not state the size of the image frames in their experiments. In another approach, Wei et al. [9] proposed an FPGA architecture that computes the Haar-like features in parallel to accelerate the implementation. In their experiments, they achieved a rate of 15 FPS for 120×120 images. Gao et al. [10] proposed another FPGA design for the Viola and Jones face detection algorithm, where the calculation of the features was parallelized for FPGA, but some necessary pre-processing and post-processing tasks were assigned to the host. They obtained a rate of 98 FPS for 256×192 image frames. In another approach [11], [12], Cho et al. proposed a complete design of the algorithm for FPGA, capable of processing 3 features, and in later versions, 8 features in parallel. They achieved frame rates of 6.5 and 16 VGA (640×480) frames per second, respectively.

While FPGAs can be very fast, particularly for well-designed digital signal processing applications, they are very expensive and often demand relatively high development time since there is a hardware configuration component involved as well. Therefore, recent attempts have been focused on accelerating the face detection algorithm using GPUs, which are massively parallel and easier to program. Harvey [13] proposed one of the first implementations of the Viola and Jones face detection method for GPU by parallelizing the feature calculation process and speeding up some of the pre-processing. Their implementation achieved a rate of 4.3 FPS for 2 NVIDIA GTX 295 GPUs on VGA-size images. In a more recent work, Hefenbrock et al. [14] presented another GPU implementation of the face detection algorithm, obtaining performance comparable to that of the FPGA implementations in [9]. More specifically, their implementation performed at about 15 FPS on a desktop server containing 4 Tesla GPUs.

Even if GPUs are cheaper and easier to program when compared to FPGA, energy constraints often prevent the successful employment of GPUs in embedded systems, since GPUs are much more power demanding than FPGAs. The implementation we propose in this paper is designed for the Gen-1 APEX family of processors, an instance of the SIMD architecture designed by CogniVue, which like GPUs, it is programmable and highly parallel, but unlike them, it is very energy efficient and has a very small footprint which makes it a perfect candidate for embedded vision solutions.

III. Face Detection Algorithm

As mentioned earlier in the paper, we use a variant of the cascade face detection algorithm proposed by Viola and Jones [1], which uses LBP features rather than Haar-like features, to produce faster and more compact classifiers [2]. We base our work on an implementation of this method and a corresponding trained classifier (i.e., `lbpcascade_frontalface.xml`) included in OpenCV.

The LBP cascade algorithm implemented within OpenCV slides a processing window over an image, evaluating successive stages of a cascade (learned by the Gentle AdaBoost algorithm) by scoring their constituent features. A feature ft describes a 3×3 neighbourhood of rectangular areas, where each of these areas has size $(ft.width \times ft.height)$. The neighbourhood, which is wholly contained within the processing window, has size $(3 \times ft.width, 3 \times ft.height)$ and its top-left corner is at an offset $(ft.x, ft.y)$ relative to the top left corner of the processing window. Figure 1 shows a schematic illustration of a feature, its parameters and how the integral image can facilitate the calculation of a feature. The value of each feature is computed by comparing the integral of the central rectangular area to that of its 8-connected neighbouring areas. This operation yields an 8-bit value called a Local Binary Pattern (*LBP*). This LBP is then used as an index into Look-Up Tables (LUTs) generated by the training process to determine whether the feature is consistent or not with the presence of a face.

A number of features together constitute a stage of the cascade. Each feature has two weights associated with it, one positive and one negative. Depending on whether the feature is consistent or inconsistent with a face, either the positive or negative weight, respectively, will be added to a sum. This sum is compared to a threshold specific to each stage; If the sum is below the threshold, the stage fails, the cascade

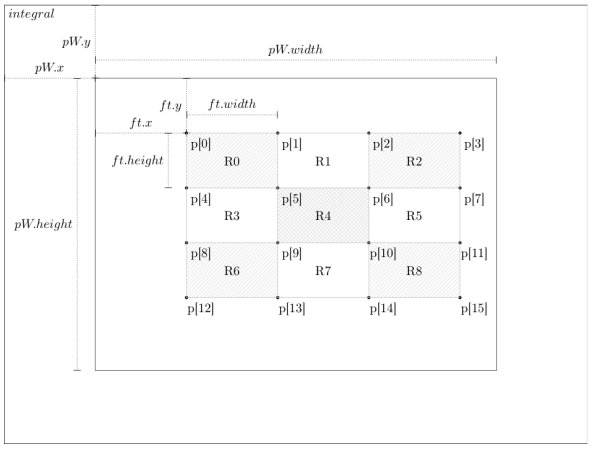

Figure 1. LBP-Multiscale Block. OpenCV's implementation uses a processing window `pW` with parameters `.w = 24` and `.h = 24`, within which features `ft` of size `.w` x `.h` at offset (`.x`, `.y`) are evaluated.

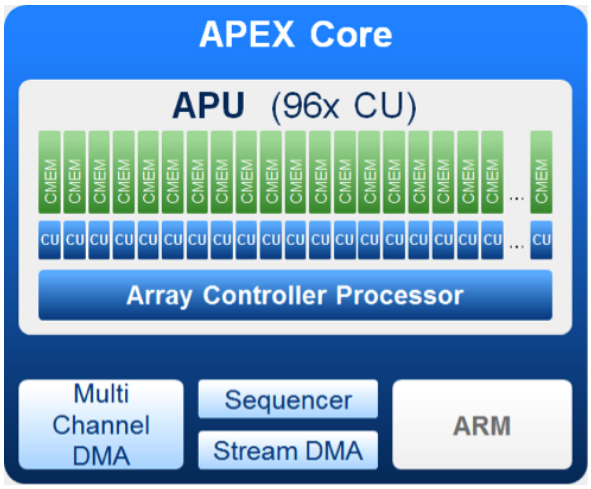

Figure 2. A schematic illustration of CogniVue's Gen-1 APEX core

terminates early, and the processing window advances to the next position. Oherwise, the next stage in the cascade is attempted. If all stages fail to reject the candidate window, it is assumed a face has been detected.

Because LBP is a single-scale classifier, the image for which to run the detection must be resized to the scales one is interested in and run the detection for each scale. This involves the computation of an image pyramid.

To avoid computing the integral of rectangles redundantly, an integral image is calculated, which dramatically speeds up the calculation of features.

IV. Implementation for SIMD

In this section, we first briefly introduce the Single Instruction Multiple Data (SIMD) architecture and discuss the limitations they pose on the implementations. We then describe our proposed implementation of the cascade LBP face detection method for SIMD designs.

A. SIMD Architecture

SIMD is a computing paradigm wherein a single instruction operates on several data points in parallel. SIMD favours uniformity of access, and therefore, data must usually be laid out in the "lanes" of a vector, and most SIMD processes perform independent operations on each lane of the operand vectors. Cross-lane operations are often limited on SIMD, usually involving only shuffles and less frequently basic arithmetic operations.

Element-wise operations on vectors are well suited for this form of processing: addition, subtraction and scaling of vectors only require element operations that are completely independent of every other element operation.

Less well suited are operations that are inherently sequential, data-driven, non-uniform or that have random data motion behaviour. Typical situations displaying this behaviour are image integration, data-driven shuffles and scatter-gather operations (storing and loading data using vectors of indices).

In this paper, we use the CV2201 SoC with one Gen-1 APEX core for our implementation. Figure 2 shows a schematic illustration of an Gen-1 APEX core. The CV2201 contains two ARM9-family processors and the Gen-1 APEX, a SIMD vector processor with 96 CUs (Compute Units). Effectively, this vector processor can operate on 96 lanes of a vector at once. However, as they are linked in a ring configuration, these CUs are restricted to accessing only their 3KB of private storage, as well as those of their immediate neighbours; More distant neighbours' storage may also be accessed but at greater cost. This places a low practical limit on the amount of data that can be simultaneously loaded into the CUs and constrains further cross-lane operations to those involving neighbouring lanes.

B. SIMD Considerations for Cascade Algorithms

Even though cascade algorithms are decidedly sequential in operation and their performance depends on early termination, there are still several ways in which data can be used in parallel or reused across feature calculations.

First, we know from Section 3 that the LBP features computed within the processing window have both an offset and a size. But it is not necessary to evaluate two features of the same size (common width and height) but different offsets at each point. Rather, we may evaluate the feature only one, cache it, and when time comes to evaluate the other, simply look up in the cache at the correct offset for the already-calculated LBP pattern. Indeed, while there are 136 features in OpenCVs cascade (i.e., `lbpcascade_frontalface.xml`), there are only 29 distinct feature sizes. As such, only 29 different LBP patterns must be calculated (one for each feature size, computed at offset $(0, 0)$). Then to evaluate a feature at point (x, y) with offset $(\Delta x, \Delta y)$, one can simply look up the pattern at

location $(x+\Delta x, y+\Delta y)$ in the buffer of LBPs as evaluated for offset $(0, 0)$.

$$LBP_{w\times h,\Delta x,\Delta y}(x, y) = LBP_{w\times h,0,0}(x + \Delta x, y + \Delta y)$$

Secondly, since SIMD architectures do not lend themselves well to divergent code paths, it may be faster to evaluate several stages of the cascade in batch, suspending rejection until the end of the stage batch. This allows for uniformity in processing at the cost of unnecessary calculations for candidates which have been rejected in the early stages of the batch.

Figure 3 illustrates the rate at which false candidates are rejected. The rejection behaviour corresponds to an exponential decay. As a rule of thumb, every 5 more stages performed leaves $10\times$ fewer candidates remaining.

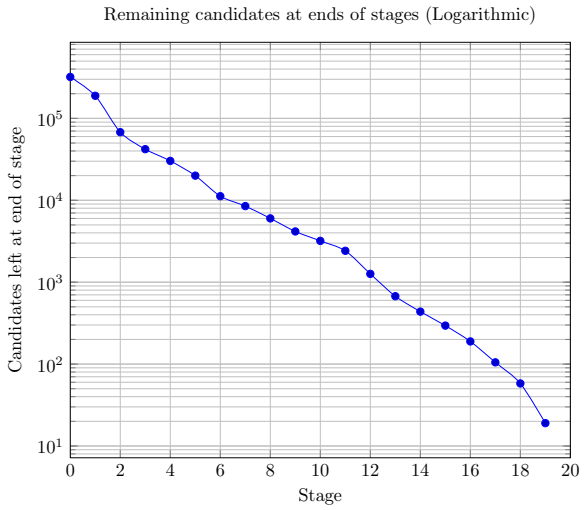

Figure 3. Rejection behaviour of the cascade

Viewed as a description of the cost of the cascade, only the computational effort below the curve is essential; Any effort above the curve is redundant because it is work done on candidates that have already been rejected.

A single-threaded CPU implementation like OpenCV's stays strictly below the curve, but it must evaluate alone all of the early stages, where there are hundreds of thousands of false candidates. This is expensive and slow.

What we propose for SIMD is therefore to stray somewhat above this curve. We perform the first few stages on the SIMD processor, performing *some* redundant work in parallel with useful work, in the expectation that the massive gain in parallelism will permit the vast majority of "easy" false candidates to be rejected in parallel faster than the CPU could have dispatched them. This frees the CPU to run only the remaining stages of the cascade on the candidates that could not be rejected by the SIMD processor.

C. Algorithm

In the classic serial LBP implementation in OpenCV, a feature's 8-bit LBP is computed on the fly from the integral image. It is then used as an index into a $2^8 = 256$-bit LUT, retrieving a single bit: 0 or 1. This bit selects one of two real-valued weights. Three to ten features are evaluated per stage, and their weights added and compared to the stage's threshold. In general, provided most features passed, the stage passes; Otherwise an early exit is made and the processing window is moved.

This implementation suffers from a number of drawbacks which affect performance, and its design makes it inherently difficult to fit in a strict SIMD processor. Firstly, the evaluation time for a single feature at a single location is in the tens of cycles. At minimum, an evaluation using this scheme requires 16 integral image reads, up to 27 integer arithmetic operations to compute the integral of the nine blocks and eight comparisons, shifts and logical ORs just to produce the 8-bit LBP. The LBP is then split into a byte offset and bit offset using a shift and a logical AND. A byte read is made using the byte offset as an index into a LUT. The bit chosen by the bit offset is then used to select one floating-point value out of two in another table, and this floating-point value is added to a floating-point sum. The sum is then compared to the threshold to determine whether or not to proceed to the next stage.

It is inconceivable that computing the LBP, using the LBP to look up into the LUT, selecting a floating-point number and adding it to the running sum could take any less than a few cycles on any architecture. Many of the steps involved are strictly dependent upon the preceding ones, and thus they constitute a long dependency chain. The latencies involved cannot be masked by pipelined, parallel or out-of-order execution.

Our implementation introduces several changes to make the algorithm suitable for SIMD architectures.

- OpenCV uses a processing window of 24×24 and thus has a maximum feature size of 8×8. All feature sizes from 1×1 through, say, 3×5 through 8×8 are permitted, making for 64 valid feature sizes.
  We instead trained a cascade with feature sizes of only 1×1, 2×2 and 4×4.
- OpenCV computes its LBPs on-the-fly, immediately discarding them once the LUT lookup is done. We on the other hand compute all LBPs before-hand and cache them. If the cascade requires computing the LBP of a given size at a given offset, the computed LBP is fetched from the appropriate buffer at the corresponding offset.
  The purpose for the restriction to only three feature sizes is that we then only require three buffers to cache the computed LBPs.
- OpenCV understandably packs its on-the-fly-computed

8-bit LBPs to a single byte to use it as an index into a LUT. But table lookups are difficult to SIMD-vectorize; Indeed only recently has the x86 architecture adopted vector gather instructions (in AVX2), and those have the limitation of being restricted to 32-bit words at their finest granularity.

We partly sidestep this problem by laying out the LBPs completely differently in our buffers. We store the LBPs in large blocks, with the bits of each LBP *striped* across it. The size of this block in bytes is equal to the width of the vector unit in bits.

We now arrive at our key leap. *The use of the 8-bit LBP as an index into a 2^8-bit LUT is equivalent to an 8-to-1 Boolean function*.

Implementing this first level of the cascade as Boolean functions may appear at first glance to have severe disadvantages. For one, we have found empirically that the Boolean functions take on average 140 AND, OR and NOT instructions to implement, and so evaluating them has very high latency.

However, even a single byte stores 8 bits, and modern processors have general-purpose registers 32 bits or 64 bits wide. Some even have vector registers that are 128 or 256 bits wide. It is thus possible, even within the simplest processors, to evaluate in n-way-parallel fashion the Boolean functions, by striping n 8-bit LBPs across 8 n-bit registers and applying bitwise operations to them. This uses only Boolean logical operations, which are the cheapest, fastest and most widely available instructions in any CPU. Often, these instructions will have a latency of only one cycle. The cost of these 140 cheap instructions is amortized across the 8, 32, 64, 128 or 256 bits of the registers in use, yielding a cost per evaluation much less than 140. Indeed, at 128 bit widths, the cost approaches $\approx$1 cycle per evaluation per pixel, much less than the serial implementation can hope to ever achieve.

This reasoning can be similarly extended to the second level of the cascade, with even more performance improvement potential. Each feature will have one of two floating-point weights associated with it, depending on the result of its evaluation. But since these weights are constants, and so is the stage threshold, it is possible to precompute all 2^n possible sums and the result of their comparison against the threshold. This yields a LUT of size 2^n bits, which can be indexed with an n-bit bitvector consisting of the single-bit outputs of the n underlying features. Again, this reduces to an n-to-1 Boolean function.

For instance, the first two stages in our cascade involve only three features each. This would still require two floating-point additions and one comparison per pixel at minimum to compute the stage's result in OpenCV's implementation. However, the pass and fail weights associated with these features are such that if any feature passes (That is, the lookup into its LUT returned a 0), the whole stage passes. If we arbitrarily decide that the value of a failed stage is 0 and a passed stage is 1, then what we need to implement the combining logic for the first and second stage is merely a 3-input NAND gate. This can be implemented as two AND instructions and one NOT instruction in software, which amortized over 128 bits amounts to the stupendously low cost of less than $\frac{1}{40}$ of a cycle per pixel.

Finally, the third level of the cascade can be viewed as a large gate which requires all stages to have reported 1 in order to accept the candidate as a face and return 1. The gate that returns 1 if and only if all its inputs are 1 is simply the AND gate.

The sum total of these points is seen in Figure 4. As the cascade progresses, a bitmap of the remaining candidates (in black) is rapidly eroded to white as the Boolean functions for each stage are evaluated and false candidates are logic-ANDed out in the third-level AND-gate.

D. Advantages

The advantages of this new approach are numerous:

- There is no data-dependent or branching code. This allows for the use of this implementation in very wide, strict SIMD, deeply-pipelined and/or superscalar processors. This approach scales linearly with the number and depth of the pipelines.
- It is easy to extend this to large vector widths. For large enough vector widths, the approach is even faster than serial code in eliminating false candidates in the early stages, and it scales linearly with the vector width.
- It is trivially and ideally suited for FPGA hardware.
- It recasts the problem of cascade classifiers into the well-studied electrical engineering problem of minimizing $n \rightarrow 1$ Boolean circuits. We have used MISII to minimize the software circuit; Little further speedup is expected from this corner.
- It depends for speed primarily on the still-increasing vector register widths, rather than on clock speeds, which have ceased increasing.

Not only is this approach well-suited to embedded systems, it is also tailor-made for the x86 architecture. Indeed, the x86 instruction set has several short-vector instructions that we may leverage to implement Boolean gates. The SSE2 instructions `PAND`, `POR`, `PXOR` and `PANDN`, available in all 64-bit x86 CPUs, can be used to evaluate Boolean functions 128 bits at a time (the width of an `XMM` register).

Newer CPUs with AVX2 can instead use the `VEX`-encoded versions of those instructions, thus exploiting the full 256-bit width of the `YMM` registers.

The x86 architecture is even blessed with *floating-point* versions of these instructions (`ANDPS`, `ORPS`, `XORPS`, `ANDNPS`). These permit even x86 CPUs without SSE2 but with SSE, or without AVX2 but with AVX extensions, to match the performance of their newer peers. For instance,

neither Intel's Sandy Bridge nor Ivy Bridge microarchitecture implements the AVX2 instruction set, unlike the newer Haswell, but both can make use the floating-point logical instructions available in AVX.

As though this was not enough, Intel's roadmap calls for the introduction in the future of the `AVX512` instruction set extension, with 32 512-bit-wide `ZMM` registers. This permits again a doubling in performance. But what's more, the `AVX512` Foundation instruction extension set includes the very useful `VPTERNLOGD`, which allows one to implement any three-operand Boolean function in a *single* instruction. The usefulness of this instruction cannot be overstated.

Aside from the ever-widening vector registers available to us, we note that Intel CPUs have long been capable of up-to-3-way out-of-order superscalar execution of register-register integer and floating-point logical operations. With latencies of a single cycle and reciprocal throughput of 0.33 cycles per instruction, Intel CPUs can effectively *triple* their already-large advantage. This underscores the scalability of our approach.

V. Experiments

In this section, we evaluate and discuss the performance of our SIMD implementation, in terms of face detection accuracy, image frames processed per second, and power consumption of the designed and developed embedded system. To verify that the modifications we made to the LBP cascade face detection algorithm to facilitate its integration into an embedded system does not result in any drop in the accuracy of the original algorithm, we ran our implementation against the CMU face detection databases Test Set A, Test Set B and Test Set C [15][16] and compared the results to that of the OpenCV implementation on the same database. Table 1 shows that restricting the available features to only those of size 1×1, 2×2 and 4×4 does not significantly impact detection. We obtain marginally more true positives on all datasets, at the price of marginally more false positives.

Database	CMU Test A		CMU Test B		CMU Test C	
# Targets	169		157		185	
Cascade	Ours	OCV	Ours	OCV	Ours	OCV
TP	121	120	90	87	153	153
FP	8	3	2	0	9	1

Table I
Face detection performance comparison

Using the system with a collection of VGA image frames, consisting of different number of faces, we observed that the system on average performs at 5 FPS. While the calculation of the LBP features is one of the fastest processes involved in the proposed face detection implementation (since LBP features are computed in parallel through the 96 CUs included in Gen-1 APEX), the progression through different stages of the cascade was among the most expensive processes. As reviewed in Section 2, quite a few embedded systems have been proposed based on FPGAs and GPUs that perform faster than our proposed system with rates of up to 16 frames per second. However, two things distinguish our work from previous ones: First, the fact that our embedded system manages to marry two normally incompatible paradigms, SIMD and cascade classifiers, through a new approach based on Boolean functions, and second, the fact that the system (i.e., CogniVue's Gen-1 APEX) has a Thermal Design Power (TDP) of only 250 mW. Compared to an FPGA design based on Virtex-5 LX330, the NVIDIA Tesla C1060 and C2050 GPUs, and the Geforce GT 220 GPU, our system consumes about 16 times, 760 times and 232 times less power, respectively. This makes our proposed system one of the most energy efficient embedded face detection systems available today.

VI. Conclusions

In this paper, we presented an embedded implementation of a face detection method based on boosted LBP features. Our proposed implementation exploits parallelism, data reuse and an entirely new approach to cascade execution in the face detection algorithm and was integrated into CogniVue's Gen-1 APEX platform, an instance of a SIMD architecture. Our experiments show that while our proposed embedded system performs sub-real-time, it is much more energy efficient than the existing embedded solutions based on FPGAs and GPUs.

References

[1] P. Viola and M. Jones, "Robust real-time face detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, 2004. 1, 2

[2] S. Liao, X. Zhu, Z. Lei, L. Zhang, and S. Li, "Learning multi-scale block local binary patterns for face recognition," in *International Conference on Biometrics (ICB)*, 2007, pp. 828–837. 1, 2

[3] "Available online at:," http://www.xilinx.com/products/designresources/powercentral/. 1

[4] "Available online at:," tp://www.nvidia.com/page/products.html. 1

[5] H. Rowley, S. Baluja, and T. Kanade, "Neural network-based face detection," in *Computer Vision and Pattern Recognition, 1996. Proceedings CVPR '96, 1996 IEEE Computer Society Conference on*, Jun 1996, pp. 203–208. 2

[6] S. Li, L. Zhu, Z. Zhang, A. Blake, H. Zhang, and H. Shum, "Statistical learning of multi-view face detection," in *Computer Vision ECCV 2002*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2002, vol. 2353, pp. 67–81. 2

[7] R. Lienhart, E. Kuranov, and V. Pisarevsky, "Empirical analysis of detection cascades of boosted classifiers for rapid object detection," in *In DAGM 25th Pattern Recognition Symposium*, 2003, pp. 297–304. 2

[8] T. Theocharides, N. Vijaykrishnan, and M. Irwin, "A parallel architecture for hardware face detection," in *Emerging VLSI Technologies and Architectures, 2006. IEEE Computer Society Annual Symposium on*, vol. 00, March 2006, pp. 2 pp.–. 2

[9] Y. Wei, X. Bing, and C. Chareonsak, "Fpga implementation of adaboost algorithm for detection of face biometrics," in *Biomedical Circuits and Systems, 2004 IEEE International Workshop on*, Dec 2004, pp. S1/6–17–20. 2

[10] C. Gao and S.-L. Lu, "Novel fpga based haar classifier face detection algorithm acceleration," in *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, Sept 2008, pp. 373–378. 2

[11] J. Cho, B. Benson, S. Mirzaei, and R. Kastner, "Parallelized architecture of multiple classifiers for face detection," in *Application-specific Systems, Architectures and Processors, 2009. ASAP 2009. 20th IEEE International Conference on*, July 2009, pp. 75–82. 2

[12] J. Cho, S. Mirzaei, J. Oberg, and R. Kastner, "Fpga-based face detection system using haar classifiers," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '09. ACM, 2009, pp. 103–112. 2

[13] J. P. Harvey, "Gpu acceleration of object classification algorithms using nvidia cuda," *Master's thesis Rochester Institute of Technology*, Sept 2009. 2

[14] D. Hefenbrock, J. Oberg, N. Thanh, R. Kastner, and S. Baden, "Accelerating viola-jones face detection to fpga-level using gpus," in *Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on*, May 2010, pp. 11–18. 2

[15] K.-K. Sung and T. Poggio, "Example-based learning for view-based human face detection," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 20, no. 1, pp. 39–51, Jan 1998. 6

[16] "Test images for the face detection task online available:," http://vasc.ri.cmu.edu/idb/images/face/frontal_images/. 6

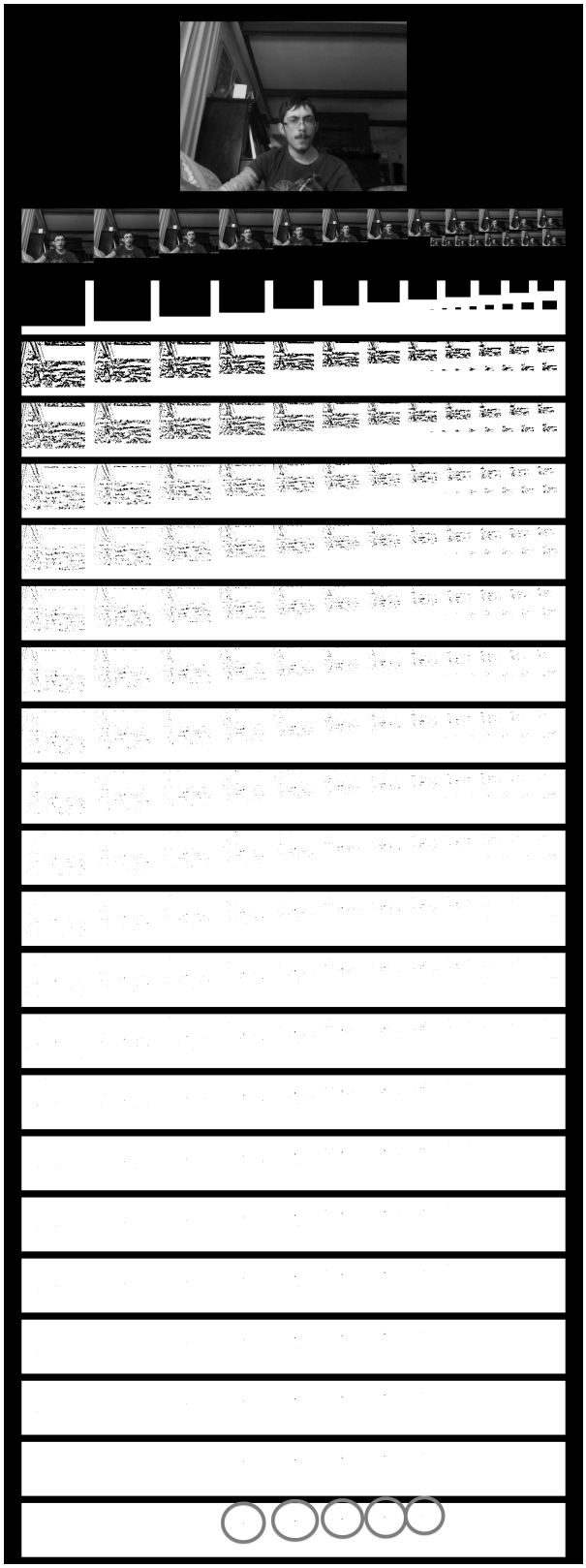

Figure 4. Detection Process. Source Image, Mipmap, Candidate Masks after each Stage. Black = Remaining candidate.