Characterising a Heterogeneous System for Person Detection in Video using Histograms of Oriented Gradients: Power vs. Speed vs. Accuracy

Calum Blair, *Student Member, IEEE,* Neil M Robertson, *Senior Member, IEEE,* and Danny Hume, *Member, IET*

***Abstract*—This paper presents a new implementation, with complete analysis, of the processing operations required in a widely-used pedestrian detection algorithm (the Histogram of Oriented Gradients detector) when run in various configurations on a heterogeneous platform suitable for use as an embedded system. The platform consists of FPGA, GPU and CPU and we detail the advantages of such an image processing system for real-time performance. We thoroughly analyse the consequent tradeoffs made between power consumption, latency and accuracy for each possible configuration. We thus demonstrate that prioritisation of each of these factors can be made by selecting a specific configuration. These separate configurations may then be changed dynamically to respond to changing priorities of a real-time system, e.g. on a moving vehicle. We compare the performance of real-time implementations of linear and kernel SVMs in HOG and evaluate the entire system against the state-of-the-art in real-time person detection. We also show that our FPGA implementation detects pedestrians more accurately than existing implementations, and that a heterogeneous configuration which performs image scaling on the GPU, and histogram extraction and classification on the FPGA, produces a good compromise between power and speed.**

***Index Terms*—FPGA, GPU, Pedestrian Detection, Histogram of Oriented Gradients**

I. Introduction

As the complexity and capability of image processing algorithms within the field of computer vision advance, the applications for these algorithms have expanded to cover embedded vision systems operating in real time: examples range from image enhancement through object detection [1] and recognition to tracking [2]. Hardware acceleration is often required to perform real-time processing [3], and this problem only becomes more acute as algorithms become more sophisticated and the volume of data to be processed grows.

A variety of COTS (commercial off-the-shelf) processors such as GPUs (general-purpose graphics processing units), FPGAs (field programmable gate arrays), and DSPs (digital signal processors) can be used for a specific processing task. Each implementation using these devices will have its own power consumption, processing latency, and result accuracy, along with other parameters such as how much processing capacity is available for other tasks. However, the energy requirements of these vision systems restrict the scenarios they can be used in. A system which can perform scene analysis tasks while balancing characteristics such as power consumption, processing time and algorithm quality (e.g. the false alarm rate of a detector) has several practical applications, especially if it can adjust these characteristics in response to changing constraints or priorities either in itself or its environment.

C. Blair and N.M. Robertson are with the Visionlab at Heriot-Watt University, UK. e-mail: cgb7@hw.ac.uk

D. Hume is with Thales Optronics, Glasgow.

Manuscript received

A. Motivation

The grand aim of our work is the development of a local situational awareness system mounted on a vehicle [4], [5]. If the vehicle is stationary with the engine off, available electrical power will be reduced, and required processing time will be relaxed. Alternatively, if the vehicle is moving or potential threats are nearby, more power for processing will be available, the frame-rate required will increase, and a lower false-alarm rate will be tolerated. In summary, we consider real scenarios where the resource constraints of a system may change dynamically. In this paper, however, we address the implications of the run-time selection of the heterogeneous architecture and its performance, for one algorithm: person detection.

Another motivating application for such a system is mobile robotics which deploy multiple sensors [6] navigating within an unfamiliar environment. A need to conserve power will usually dominate but there may be occasions where rapid or accurate processing (e.g. identification of fast-moving objects) will be required. It is important to understand what implications re-prioritising in this way has on, e.g., accuracy of detection. Hence, a vital step in building a system like this is an understanding of the tradeoffs involved when selecting one processing architecture over another, how these change if heterogeneous processors are used, and how these tradeoffs affect application performance.

More broadly, in any resource-restricted mobile surveillance system, the scene currently observed by a camera will dictate the kind of processing required at any given time. This may include enhancement of regions, detection of objects or deployment of other sensing modalities. If timeliness or low power is also a priority, we must choose which algorithm to run and the processor to run it on. This becomes a *tradeoff* between availability of a processor, accuracy required, power available and latency.

In this paper, we use a pedestrian detection scenario as the application to investigate this question. We are partly

motivated by its importance in many real applications (service or factory robotics [7], or advanced driver assistance in vehicles [8]) and partly by the fact that person detection is a core component of our own, larger system.

We note that advances in pedestrian detection are being constantly made [9], [10] and we choose HOG for the following reasons: a. it remains the most popular open-source detector, with implementations available for various accelerated processors; b. new algorithms [11] rely on histograms of gradients and other image channels; c. there are baselines in performance on FPGA/GPU/CPU (see §II-B1) for which we may compare our implementation.

We note that the evolution of computer vision is moving quickly towards widespread use of GPU implementations [12] and the use of CPU/GPU combinations. There is a good argument therefore to include FPGA since it has been demonstrated that image processing operations are amenable to considerable speed enhancements on this architecture [13]. In this work we exploit FPGAs, GPUs and a combination of both (as well as CPU) in order to perform real-time processing in this task (see Figure 1). These architectures differ in their memory hierarchy, approach to parallelism (both spatial and temporal), and compute to input/output performance. FPGAs also differ from platforms with a software design flow (processors and GPUs) in that implementation requires a more involved understanding of the hardware.

The current literature focusses on implementations of algorithms for a specific platform, or comparisons of execution time of more simplistic operations between devices. However, there is a lack of a detailed comparison for larger and more complex algorithms, and that is what this paper addresses.

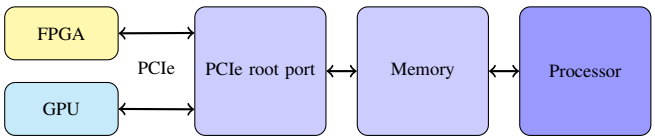


Fig. 1. System Architecture showing the interaction for the GPU, FPGA and CPU. Both accelerators can access main memory through DMA.

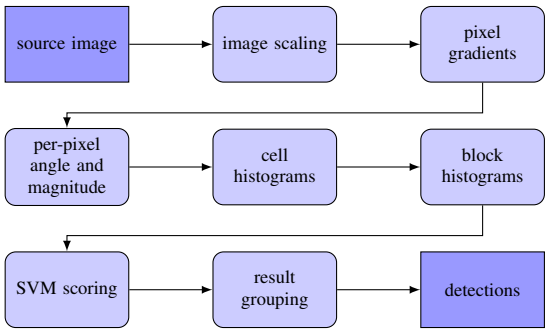


Fig. 2. HOG algorithm stages: (i) image scaling; (ii) gradients; (iii) angle & magnitude calculation; (iv) generation of oriented histograms over cells; (v) block concatenation and normalisation; (vi) linear SVM scoring; (vii) result grouping.

B. Contributions

In this work we describe a platform with multiple heterogeneous accelerators and use the HOG pedestrian detector as the example application to investigate the system's characteristics and performance. A schematic of the algorithm is shown in Figure 2. The platform contains a FPGA, GPU and GPP (general purpose processor or CPU) (again, the schematic is shown in Figure 1).

The HOG algorithm is partitioned between processors in several new ways. Stages are run on one processor or another, or a combination of all three, with different characteristics in each case.

In summary the contributions of this work are:

1) We describe the performance characteristics (power, speed and accuracy) of various algorithm execution paths using different combinations of accelerators.
2) We quantify the potential benefits available when using a combination of accelerators in a single execution path, the performance tradeoffs seen when mapping algorithm operations onto a particular accelerator, and the aggregate effect on performance once communications delays are taken into consideration.
3) If a particular performance characteristic is prioritised, we show *which configuration should be selected and the tradeoffs made when doing so.*
4) We do this with the goal of extrapolating these results to other compute-intensive image processing algorithms which use similar detection and classification techniques.
5) In addition to this heterogeneous system, we also document our implementation of HOG on an FPGA which to our knowledge is the most accurate available for large image sizes.

C. Paper Roadmap

Section II describes the background and related work, including examples of applications where flexible processing platforms are an advantage, previous architectural comparisons and analyses, and methods for automatic decisions for allocating operations to architectures. Section III covers the implementation, including the platform, a description of the algorithm used and modifications made to it to suit the underlying architecture. Results and a comparison to existing implementations are given in section IV. We draw conclusions from this, describe the tradeoffs between configurations, and describe our plans for future work in section V.

II. Related Work

Existing work which motivates this work and places it in context is given in section II-A. Pedestrian detection algorithms and existing implementations are covered in §II-B. In section II-C, architectural differences and work on performance comparisons are discussed.

Many studies show performance gained when moving certain processing operations to an accelerator, and application speed-ups of tens to hundreds of times are regularly reported. Several studies also compare the performance of different accelerators for various elementary operations (e.g. 2-D image filtering) [14], and show a trade-off between performance

in various accelerator architectures at various kernel sizes. Designers can use this information to choose an architecture to use to accelerate a specific algorithm. Alternatively, tools exist which perform design space exploration automatically [15] and allow the hardware engineer to select from various Pareto-efficient area vs. latency tradeoffs when designing an ASIC.

A. Context-dependent Processing

The field of surveillance and security is one in which advances in computer vision algorithms and platforms can quickly be applied to existing problems. Consider Letham et al. [4] as an example: segmentation algorithms are used on visual and infrared sources to label areas of an image containing road, sky and different types of vegetation. This information is then used to improve the accuracy of classifiers on that data, and elsewhere [16] to improve pedestrian detection accuracy by taking the contextual information surrounding a detection into account (i.e. we expect to find a pedestrian on the road, but do not expect to find them in the sky). False alarms are therefore reduced. This technique relies on the framework from Wolf and Bileschi [17], which samples locations at distances around a possible detection to improve the context (e.g. building, road, tree, sky) which that detection is placed in. Letham et al.'s technique relies on fixed regional classifiers, specific to the environment (a rural, open and wooded environment); use of this in other environments would require the use of other scene segmentation techniques. In an embedded system, these would require to be running already or to be loaded as required, e.g. by reconfiguration of an FPGA.

As an example of context-dependent sensor allocation, Matzka [18] expresses the problem as one of optimisation. In a car moving in traffic, the question of which cameras or sensors (i.e. scene regions) to direct attention and hence processing to is dictated by a combination of five factors. These are regions which: have high saliency, have a low time-to-collision, have vulnerable traffic participants, have participants which can be classified in the time available, and, when observed, cause a large reduction in uncertainty. This is a specific example of context-dependent sensing in more well-known tasks, such as autonomous vehicle navigation [19] where one or more of visual and laser data is used for road finding at different distances and speeds.

B. Pedestrian Detection

Pedestrian detection is one of the most common object-detection tasks with both a real-world and real-time application, and enough algorithms have been described to allow comprehensive comparisons to be made [8], [9]. One of the most popular in terms of existing implementations is HOG (histogram of oriented gradients). Dalal and Triggs' original paper [20] was included in a review of pedestrian detection: Dollár et al. [9] rank HOG as having middling performance for computation time and accuracy compared to alternative methods. In addition, almost all the detectors Dollár evaluated used gradient histograms, so the techniques described in this paper would also apply there.

1) HOG Implementations: HOG has been implemented for GPU [21], in a FPGA-GPU system [22], and also either partially or fully on FPGA. Kadota et al. [23] perform HOG feature extraction in FPGA, then classify the results on a microprocessor. Martelli et al. [24] perform FPGA-based pedestrian detection using covariance features, and Hiromoto et al. [25] describe a similar system using co-occurrence HoG. We compare our implementation to these versions in the results. Similar techniques involving histograms of local features followed by SVM classification have been used to perform detection of vehicle orientation [26] and road signs [27].

C. Architecture Choice

Several direct architectural comparisons between different COTS accelerator architectures which focus on performance have been done. Early work focused solely on fundamental operations such as matrix multiplication of varying sizes; this gave way to domain-specific comparisons such as optical flow [3]. Cope et al. [14] perform a broader FPGA/GPU comparison and conclude that an algorithm's memory access rate and patterns can greatly affect the performance of any implementation. Selection of an accelerator for a particular application must therefore be done with this in mind.

Operator Allocation Strategies: In addition to the choice of accelerator architecture, we must decide which stages or operators of an algorithm to accelerate. For this operator mapping problem, two approaches are possible; a designer can chose an accelerator based on prior knowledge of the algorithm and architecture characteristics, or this decision can be made algorithmically.

In an example of the former, a static detection system based on HOG, Bauer et al. [22] perform hypothesis generation by calculating gradient histograms on FPGA, then doing kernel SVM classification of regions of interest on GPU, thus overcoming the limitations inherent in the FPGA architecture.

Quinn et al.'s work. [28] is an example of the latter approach to partitioning. The Dynamo system is a hardware-software platform which reconfigures itself in response to requests to apply a certain sequence of operations or processing pipeline to an image. The pipeline is made up of blocks for which resource use (in the form of coprocessor area) and latency are already known. The innovative step here involves decision making at runtime; the system performs multiobjective optimisation to simulate various hardware/software algorithm arrangements using dynamic programming (a multiobjective optimisation algorithm) to select the lowest-latency partitioning scheme that fits within the area available on the reconfigurable hardware. The FPGA is then reprogrammed and the task is rerun, in a process which takes several seconds. Although the decision-making is performed at runtime, this system is task-reconfigurable but is unable to react to changes faster than this.

The Encore system described by Zuluaga et al. [15] is another example of static hardware/software partitioning. This analyses the dataflow paths in an algorithm, calculates the latency and area improvement produced by moving certain instructions from software to hardware, and produces a Pareto

curve allowing the designer to select a particular partitioning arrangement, based on the desired area/latency tradeoff. Thus, the decisions of which accelerators to use and which algorithm stages to accelerate can be made programmatically at runtime.

III. Implementation

Section III-A describes our platform architecture. A description of the HOG algorithm follows in section III-B. Based on this, we partition the algorithm between devices as described in III-C. Our FPGA architecture for HOG follows in III-D. An alternative classification method is discussed in III-E and classifier training is covered in III-F.

A. Platform

Our platform, shown in Figure 1, is based on a dual-core Intel Xeon 2.4GHz CPU, a NVIDIA GeForce 560Ti GPU and a Xilinx ML605 board containing a Virtex-6 XC6VLX240T FPGA. Both accelerators are connected over PCI-express. A PCIe 2.0 x16 link is used for the GPU, and a 2.0 x4 link for the ML605. The GPU requires both a PCIe link and a host x86 CPU, so a PCIe link was used for data transfer to and from FPGA. The use of a host CPU allows the FPGA to be treated as an accelerator to an existing processor rather than a standalone streaming processor. This approach simplifies the FPGA control logic and also allows loading of frames via the host, either from camera or video, or network. This removes the need for a separate framegrabber connected to the FPGA.

B. HOG Algorithm

The histogram of oriented gradients algorithm was originally targeted at pedestrian detection but can be used to detect other objects. This is a description of the original algorithm from Dalal's work [20]. A block diagram is shown in Figure 2. Here we describe the complete algorithm, then later discuss the steps which must be modified to perform the calculations on a FPGA.

Algorithm Stages:

i. *Gradients* First, the colour source image is split up into windows of 128 vertical by 64 horizontal pixels, each containing blocks of 16×16 pixels, which in turn are made up of four 8×8 pixel cells. Each colour channel is then processed separately. In each case, the source pixels belonging to each cell are gamma-corrected by taking the square root, then convolved with a $[\ 1 \quad 0 \quad -1\ ]$ kernel in both dimensions to generate gradients g_x and g_y.

ii. *Orientation & Magnitude* For each pixel in the cell, magnitude $M = \sqrt{g_x^2 + g_y^2}$ and $\arctan \frac{g_y}{g_x}$ are calculated. Dalal used 9 bins, split over $0 - 180°$; he noted that this is effective for pedestrian detection, but can be changed to $0 - 360°$, e.g. for vehicle detection. Constants $b_0, b_1, \ldots, b_9$ representing the edges of each bin are set as $\tan 0°, \tan 20°, \ldots, \tan 180°$. For each pixel, $\frac{g_y}{g_x}$ is evaluated and e.g. bin B_0 is chosen if $b_0 \leq (g_y/g_x) < b_1$.

iii. *Histogram Generation* M is then weighted based on the difference between the gradient angle and the angle of the bin edge, and added to the eight surrounding bins (bins B_n and B_{n+1} in all four cells in a block); this prevents quantisation errors caused by large-magnitude weights close to a bin edge. These steps produce a 1×36 block histogram vector bv.

iv. *Block Normalisation and Concatenation* This vector is then normalised to produce bn in two stages: $bn' = \frac{bv}{\sqrt{|bv|_2^2 + e}}$, where $e \ll 1$. bn' is capped at 0.2, then $bn = \frac{bn'}{\sqrt{|bn'|_2^2 + e}}$. All the block vectors in the window are concatenated, producing a feature descriptor fv. These descriptors are produced for 7 horizontal by 15 vertical overlapping block histograms which make up the 128×64 pixel window.

v. *Window Scoring* This feature descriptor is multiplied by SVM weights of 7×15 elements with 36 weights each. This produces a score $s = \sum_{n=1}^{3780}(fv_n \cdot w_n) + b$ for the window.

vi. *Image Shifting and Scaling* HOG is a sliding-window detector; the steps above are performed on a window offset by 8 pixels from the previous one. This overlapping allows better detection, but a pixel and hence a block may belong to up to 105 windows and calculations will be duplicated. In practice, block histograms for the entire image are calculated, then the classifier window is slid over the result. Scoring is repeated over all windows, then the image is downscaled and the process is repeated several times per octave.

vii. *Result Grouping* Finally, the scores for all windows at all scales are then aggregated using mean shift and a pedestrian detection is registered if this aggregate score is above a given threshold. This is done using OpenCV's default grouping algorithm.

TABLE I
Data dimensions for each algorithm stage, per 1024×768 frame

Algorithm Stage	Single-scale data (kB)
source	768
cell histograms	432
normalised blocks	1670
window scores	38

The CPU and GPU-versions of these stages are provided by OpenCV code. The OpenCV v-2.4 CUDA implementation [12] corresponds closely to the original algorithm.

C. Algorithm Partitioning

We now consider algorithm partitioning and hence possible dataflow paths. Similarly to §II-C, we investigated various algorithm acceleration strategies between the host processor, GPU and FPGA, taking into account the processing operations and size of intermediate data generated at each step. Algorithm stages i–iii mentioned in §III-B are stream processing and can be performed on any architecture, but map well to a heavily-pipelined FPGA. Stages iv–v involve vector multiplications. These map easily to GPUs and multicore processors, and are well-suited to the dense arrays of floating-point multipliers on the GPU. Memory access patterns at this stage change from

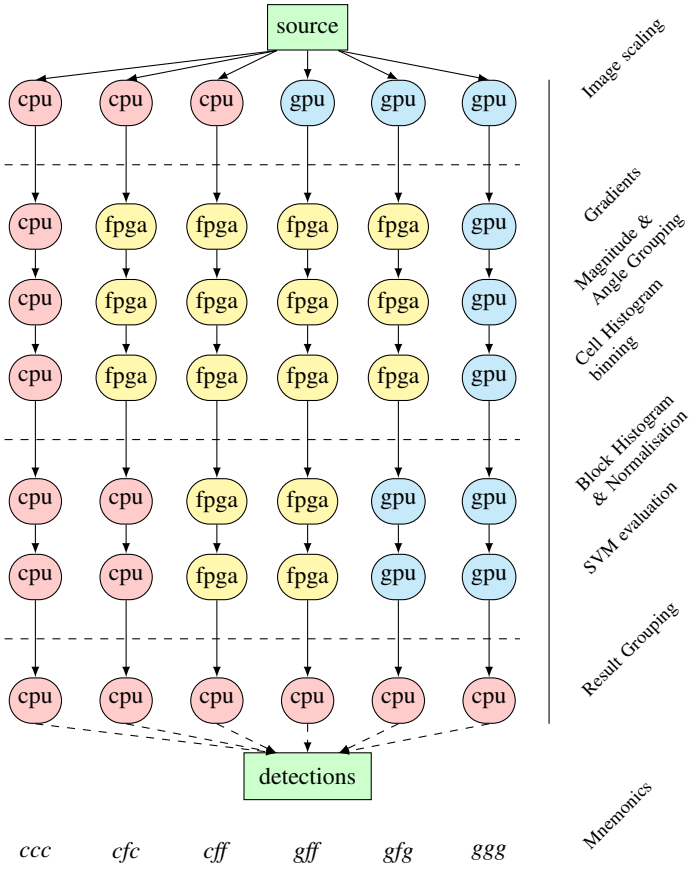


Fig. 3. Six possible processing paths through the algorithm. Any one of these can be selected and used to generate detections. We define these using the three-letter mnemonics above, and refer to them throughout the paper.

pixels which can be read for a few lines then discarded, to cell histograms which must be copied to multiple blocks, and classifier weights which are used many times over. A firmware architecture for these stages is more resource-intensive and also requires more development time. Finally, result grouping operates on much smaller volumes of data so in all cases we perform this on the host PC.

Considering the data dimensions at each step and realising that data transfer between different devices between stages will incur communication delays, we can generate a list of dataflows through the application which are viable candidates for implementation (shown in Figure 3). Based on Table I, the only data which can reasonably be transferred between different processors is input pixels at our chosen scale, cell histograms or window scores. We split these paths into 3 stages: scaling, cell histogram generation, and classification, and refer to them by the processor used to perform each task, e.g. *gfg* means "scale on the GPU, generate cell histograms on the FPGA, then normalise and classify on the GPU". Specifically, we wish to investigate whether mapping processing of the early stages to FPGA and doing block processing on GPU outperforms processing the entire algorithm on an individual accelerator.

D. FPGA HOG Architecture

The FPGA architecture is made up of stripe processors placed side-by-side (see Figure 4. 16 stripes are required for a 1024×768 image). Each one operates on a 64-pixel wide stripe of image data and generates cell and block histograms for all pixels within it.

All calculations involving a cell are performed as pixel data is streamed in. These are detailed in §III-D1 and Figure 4a. If the data is then being transferred to the host or GPU, the cell histograms can then be read out and discarded from the FPGA. Alternatively, §III-D2 and Figure 4b describe classification on FPGA. For multiscale evaluation at n scales, the frame is scaled on the GPU or host CPU then padded to the original image size and passed to the FPGA for processing. While this involves more data transfer and is slower than on-board image scaling, complexity is reduced, particularly if many scales per octave are needed.

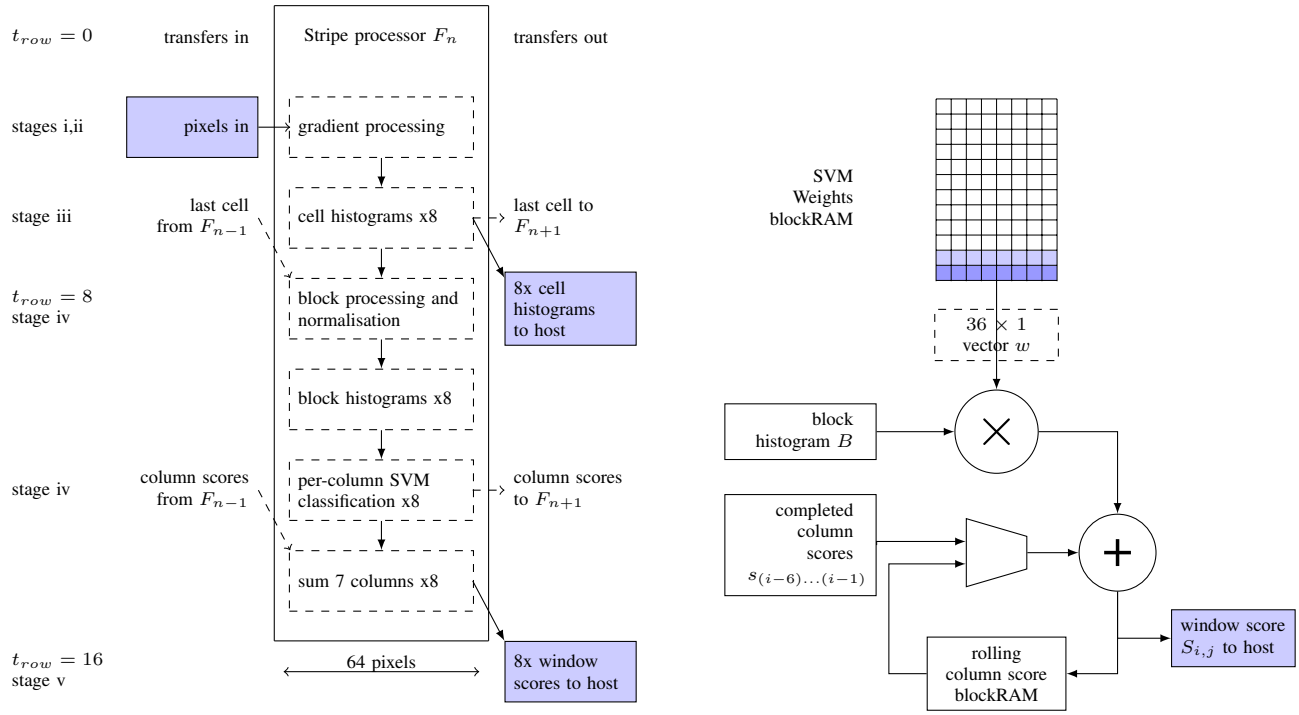
1) Cell Histogram Operations:

Orientation and Magnitude:
Due to limitations in our PCIe interface we convert the colour image to grayscale before transfer to the FPGA. For magnitude generation, we use the magnitude approximation described in Wilson et al. [29] ($M_{approx} = \frac{1}{1+\sqrt{2}}(|g_x| + |g_y| + \sqrt{2} \times max(|g_x|, |g_y|))$) to avoid square-roots. We then select an orientation bin without using division or trigonometric calculations, using the method described in Bauer et al. [30]: we retain constants $b_0, b_1, ..., b_9$ then set $g_x = -g_x$ and $g_y = -g_y$ if $g_x < 0$, then select bin B_0 if $b_0 g_x \leq g_y < b_1 g_x$ etc.

Cell Histogram:
Once an angle bin for a pixel is selected, the pixel's magnitude is then added to the relevant bin for that cell. Weighted voting into several angle bins is omitted; this simplifies the calculation and allows re-use of cell histograms in adjacent blocks. These histograms are stored in accumulators and, after every 8 rows, are either read out to host memory or passed to addressable shift registers (ASRs) within the block normaliser (stage iv in Figure 4a). As each stripe operates on 8 cells, we need to store ten cell histograms (eight from row i and two from $i-1$) in the ASRs at any one time to generate a block histogram.

2) Window Classification Operations:

Block Histogram:
All cell histograms in each stripe are loaded from the ASRs filled in the previous stage, and then normalised. Some of the overlapping blocks span two stripes so cells are shared between stripes when necessary, as shown by the transfers in Figure 4a. The normalisation logic is shared between all blocks in a stripe due to its complexity. The L1-norm is taken instead of the capped L2-norm: $bn = \frac{bv}{\sqrt{|bv|+e}}$. This only requires one division and square root, instead of two square-roots, divisions and dot products.



(a) HOG Stripe processor

(b) C_i, one of eight block classifiers in a stripe

Fig. 4. HOG stripe processors extract histograms from cells sequentially. A stripe processor (a) shares the normalisation logic, classification control logic and SVM weights between its block classifiers. As pixels are fed in, cell histograms are generated, stored in an addressable shift register, and read out after 8 rows. Internally, these are then concatenated and normalised into block histograms. Cells from neighbouring stripes are included for blocks which overlap between stripes. (b) At stage iv, each block *B* is classified for all windows it belongs to and the partial scores accumulated. Once a column is complete, column scores from the rest of the windows are added to produce a window score.

SVM evaluation:

For histogram classification on the FPGA, 16 rows of cells must be retained for each window of $7 \times 15 = 105$ blocks; this is an impractical amount of data to store. We avoid this problem by normalising the cells into the appropriate block histograms B, immediately classifying that block against each of the 105 overlapping windows it belongs to, then discarding the block histogram and retaining only 105 partial sums. This is done in the block classifiers within each stripe (Figure 4b). For a window at location $m_{i,j}$, 7 partial sums $s_{i-6}, s_{i-5}, \ldots, s_i$ representing columns of blocks $m_{i-6}, \ldots, m_i$ will be stored in the blockRAMs of classifiers $C_{i-6}, \ldots, C_i$. These will be updated as each new row of blocks is processed. Once all rows forming window $m_{i,j}$ have been processed, s_i will contain partial sums for blocks $B_{i,(j-15,j-14,\ldots,j)} \cdot w$ and so on. Columns $s_{i-6,\ldots,i}$ are summed in C_i to form a window score $S_{i,j}$, which is then transferred to the host. However, because the sliding windows overlap vertically and horizontally, the RAM in C_i also contains partial sums for all windows which contain the location $m_{i,j}$. As before, these are read out as each window finishes, and transferred from the preceding stripe F_{n-1} if necessary. Thus, instead of sliding an image window through a classifier, we evaluate all elements in the support vector for each new block then gradually sum the results as new blocks are presented. This requires one blockRAM and one embedded multiplier for each column of blocks in a stripe. Normalisation of 8 blocks per stripe and the 3780 multiplications required by each block must be completed before the next cell histograms arrive; this prevents the FPGA version from being scaled down to smaller image widths, but can work on larger image widths by adding more stripes.

These optimisations, while making processing feasible, degrade the accuracy of the FPGA-based versions by a few percent, as shown in §IV-B.

3) Operations for cfc and gfg: The same cell histograms which are used on FPGA are transferred back to the host and then to optionally to the GPU. Here, they are processed in the same way as the original GPU data. We calculate full L2-norms and use different classifier weights to account for the earlier algorithm differences. The FPGA logic was designed in a Simulink System Generator model and then synthesised using ISE 13.4. All FPGA versions use the same bitstream, with data being extracted at either the histogram or score stage. This means that reconfiguration of the FPGA between selection of different versions is not required.

E. SVM Radial Basis Function Classification

Linear evaluation of a feature vector **fv**, involves finding $s = \sum_{i=1}^{3780} (fv_i \cdot w_i) + b$. A SVM using a radial basis function (RBF) kernel is an alternative to this, and increases accuracy

at the expense of runtime. In this case, $s = \sum_{i=1}^{n_{sv}}(\alpha_i \cdot y_i \cdot \mathbf{K}(\mathbf{fv_i}, \mathbf{x_i})) + b$, where $\mathbf{K(fv,x)} = \exp(-\gamma\|fv - x\|^2)$. We implemented RBF classification on the GPU for *gfg* and *ggg* versions. They are included in the results in §IV as *gfg-kernel* and *ggg-kernel* respectively.

We also calculated the runtime for FPGA RBF classification but did not implement it. See §V-A1 for a discussion.

F. Classifier Training

Classifiers were trained on the positive images and a sample of the negative images in the INRIA training set, then re-trained once on negative errors using SVMLight. Two sets of classifier weights were generated, one for *cfc* and *gfg* and one for *cff* and *gff*. The *ggg* and *ccc* versions use the weights from [20]. RBF classifiers were trained in the same manner.

IV. Results

We now compare execution time, accuracy and power consumption of each implementation. During execution it was possible to change processing from one processing path to another in consecutive frames with no delay due to switching.

Dollár notes that HOG detects pedestrians at around 90 pixels in height, placing them at 20m or more away from the camera [9]. Scaling allows pedestrians much closer than this to be detected, at the expense of detection runtime. In our application we are more concerned with detection at a distance (bearing in mind we are interested in detection from a mounted vehicle), thus we evaluate performance at $n = (1, 3, 13, 37)$ scale levels, scaling the image by $1.05\times$ between each level.

A. Performance Considerations

Using this system, we performed pedestrian detection on a $w = 1024 \times h = 768$ video at our chosen scale levels ($n = 37$ is the maximum number of scales for this size of video). Overall processing times for each version are given in Table III. From these, the FPGA versions are fastest, taking $4.88ms$ at $n = 1$ (*cff* and *gff* are identical as no scaling is done at this level). In these cases, most of the time taken is spent moving the image data through the FPGA (1024×768 pixels at $200Mhz = 3.91ms$). A heterogeneous system only shows a slight speed advantage at 3 scales, while for $n > 3$ the *ggg* version is consistently faster.

Figure 5 shows processing times of individual algorithm stages and transfers between them. The FPGA implementation is not capable of performing the multiple image scaling per octave that HOG requires for accurate detection; this has been mitigated by rescaling on the CPU or GPU and padding and transferring the result. Due to limitations of our PCIe interface, each scaled image must be transferred separately and incurs its own overhead, in the form of additional data transferred after the frame data to push remaining window scores or cell histograms through the pipeline. For multiple scales, this means the pipeline is emptied between each scaled image. The "FPGA histograms" and "FPGA scores" bars in Figure 5 also include transfer to and from the FPGA. Data is transferred via DMA into buffers in blockRAM on the FPGA. These buffers are refilled as the processing logic consumes pixel data. Unlike on the GPU, this approach means there is no discrete "data transfer to or from the FPGA" step; after the buffer is filled initially, transfer time is hidden and is dictated by the rate at which the HOG logic consumes and generates data.

The GPU implementation avoids this problem by storing each frame in its own global memory, performing on-board scaling, and only transferring individual detections back to main memory; the compute:i/o ratio is higher.

The inefficiencies associated with the multiple transfers to FPGA mean that all FPGA versions run slower than *ggg* at multiple scales. In addition, for *gfg* the cell histograms must be transferred twice (from the FPGA back to main memory then to the GPU). However, the PCI-express specification allows for direct endpoint-to-endpoint transfer, allowing an FPGA to transfer data directly into GPU on-board memory. This technique is currently only possible with Kepler-class GPUs on Linux hosts [31] but is an option to consider for future work.

Power consumption of the whole PC system for each version is shown in Table IV. The bottom half of this table compares each accelerated method. Consumption in *gfg* and *ggg* appears to increase with the number of scales, while power for the versions where the FPGA does most of the processing remains constant. There is a tradeoff around $n = 3$ again; above this, FPGA-based systems draw less power. In a system where all 3 processors are switched on, the *cff* or *cfc* versions draw the least power at all scales. When implemented, all FPGA versions ran at $f = 200MHz$. Implementation details are given in Table II.

TABLE II
Resource Utilisation for VLX240 FPGA for HOG application and PCIe link logic

Resource	Capacity
Registers	40%
LUT	72%
Slice	92%
BlockRAMs	25%
Embedded Multipliers	18%
Overall	72%

TABLE III
Processing times for each execution path (Labels defined in Figure 3)

Linear SVM	Time (ms)			
	1 level	3 levels	13 levels	37 levels
ccc	172	480	1365	1826
cfc	72.9	226	973	3328
cff	**4.88**	22.6	103	233
gff	**4.88**	**17.4**	78.5	196
gfg	7.82	25.4	111	303
ggg	7.10	17.8	**47.6**	**59.0**
Kernel SVM				
gfg	2200	6540	28280	81180
ggg	3540	9570	25700	35345

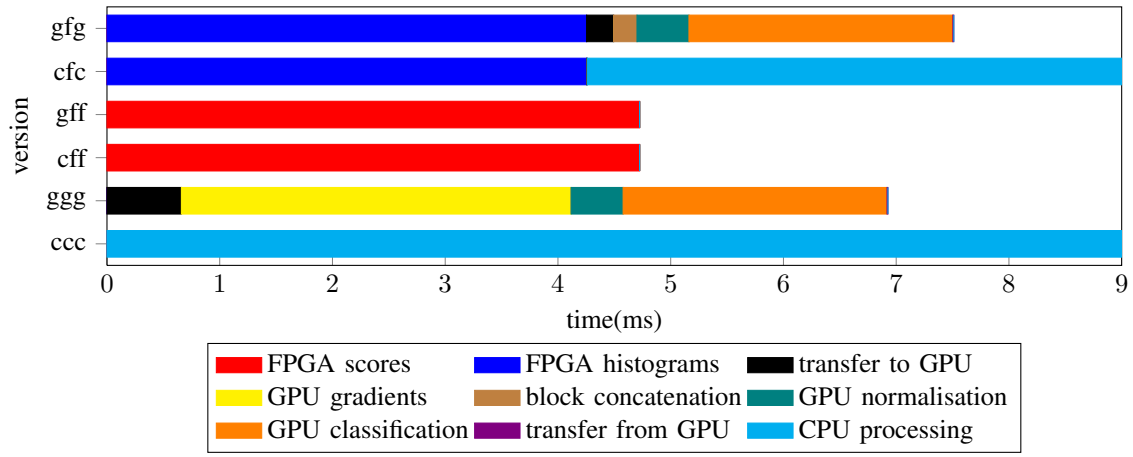


Fig. 5. Time (in ms) spent on each algorithm stage for each version, at $n = 1$ (no scaling). Transfers to and from the FPGA are contained within the FPGA measurements, which also includes extra non-image data transferred to flush the buffer.

TABLE IV
SYSTEM POWER CONSUMPTION FOR EACH EXECUTION PATH AT 1, 3, 13 AND 37 SCALING LEVELS

Version		Power Consumption (W)			
	FPGA	1 level	3 levels	13 levels	37 levels
cpu/gpu idle	off	129			
	on	147			
ccc	off	156	155	156	161
ccc	on	179	179	179	182
ggg	off	170	180	198	201
cfc	on	183	**177**	**176**	**184**
cff	on	**179**	**177**	178	**184**
gff	on	182	181	180	**184**
gfg	on	187	193	201	206
ggg	on	194	205	221	227

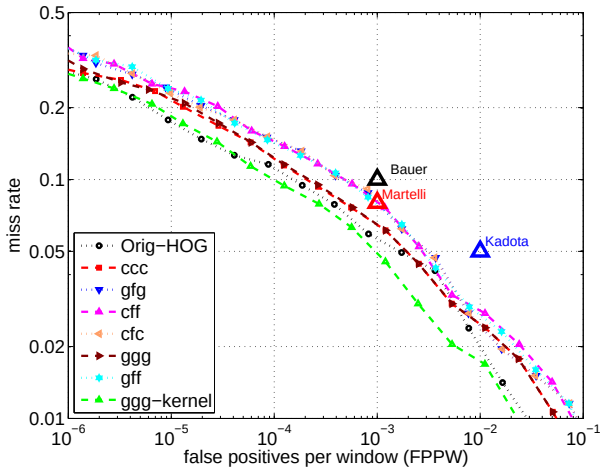


Fig. 6. DET curve for single-scale HOG on INRIA dataset

B. Detection Performance

Figure 6 shows a detection error tradeoff (DET) curve for the INRIA pedestrian dataset. All images are padded to 1024×768 before evaluation. The results are comparable to the original, with a slight decrease in accuracy in the *gfg* and *cfc*-versions due to simplified block weighting, and a further decrease for *gff* and *cff* due to the simplified normalisation. This figure also shows comparisons to other FPGA implementations, showing that our implementation is more accurate. Figure 7 compares each of our implementations to the performance of the original HOG algorithm on the large positive test set from INRIA, using the evaluation code from [9]; this allows comparison to other algorithms and implementations. On this graph the differences between our versions are more pronounced. *cfc* and *gfg* versions are still similar.

C. Comparison to State-of-the-art

Results from the 3 FPGA implementations cited in §II are plotted in Figure 6. Kadota et al. [23] perform HOG feature extraction on FPGA at 30fps on VGA video, with 5% miss rate on the INRIA dataset at 10^{-2} FPPW. Martelli et al. [24]

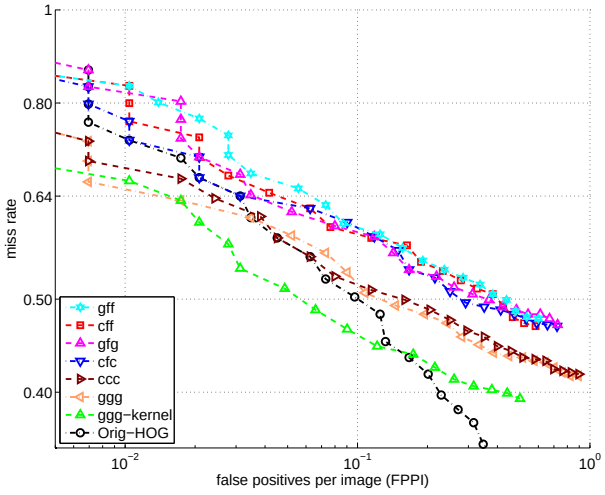


Fig. 7. DET curve for multiscale HOG on INRIA dataset

TABLE V
CHOICE OF ALGORITHM VERSION AND COMPROMISES FOR A GIVEN PRIORITY AT 13 SCALES. TRADEOFFS SHOWN AS PERCENTAGE DIFFERENCES FROM THEIR BEST MEASUREMENT. ACCURACY IS MEASURED AS % CHANGE AT 10^{-4} FPPW.

Priority	Version to use	Tradeoffs		
high speed	*ggg*	power ↑ 26%	accuracy ↓ 3%	
lowest power	*cfc*	time ↑ 2000%	accuracy ↓ 3%	
high accuracy	*ggg-kernel*	time ↑ 54200%	power ↑ 26%	
power and speed	*gff*	power ↑ 1%	time ↑ 65%	accuracy ↓ 5%

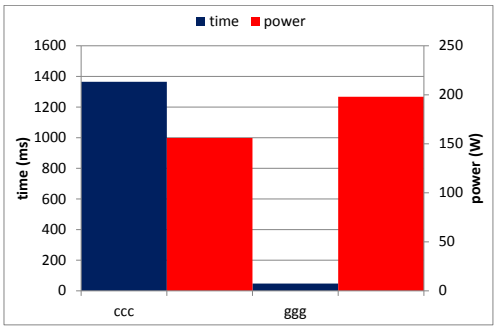


Fig. 8. Power and time measurements for *ccc* and *ggg*. These offer two extremes of power and speed, and we compare heterogeneous configurations to these as a baseline.

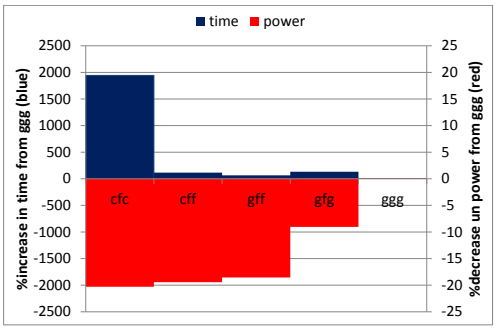


Fig. 11. Relative change in power consumption and processing time of each configuration compared to *ggg*. A larger red bar indicates decreased power consumption.

perform FPGA-based pedestrian detection using covariance features, achieving 20% miss rate at 10^{-4} FPPW on INRIA. As shown in Figure 6, all our implementations including *cff* and *gff* outperform these. Hiromoto et al. [25] describe a similar system which works on co-occurrence HoG. They do not provide accuracy information, but evaluate a QVGA image at scale ratio $s = 1.2$ with 3615 windows per frame at 38fps or 139166 sub-windows per second, whereas our *gff* version described above evaluates 20868 windows per frame at 13fps (271284 sub-windows per second) for the same s. Kadota et al. [23]'s implementation evaluates 56466 windows on 10 parallel elements, taking $5.7\mu s$ per window. In contrast, our FPGA version takes $657\mu s$ to evaluate histograms over a single-scale window or a further $40\mu s$ to generate window scores, using up to 121 parallel elements to generate a row of scores across the image at once. This longer window period is dictated by the pixel clock and our larger frame width. In both cases, our slower framerate is a limitation of our PCIe architecture; if the multiscale evaluation was fully pipelined, multiscale evaluation at $s = 1.2$ would take around $20ms$.

V. DISCUSSION

Our aims with this work were: first, to investigate the performance of a platform using heterogeneous architecture when applied to a real-world image processing problem; and second, to evaluate the tradeoffs between our implementations.

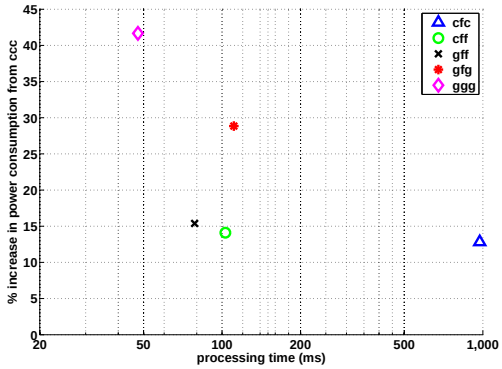
A. Tradeoffs

Based on the power, latency and accuracy results in Figure 6 and Tables III,IV, we can now give specific tradeoffs for this system, and evaluate the costs of selecting one configuration over any other. This is summarised in Table V for $n = 13$. In the remainder of this section, we consider the viable implementations for pedestrian detection in this system. We discuss the *ggg-kernel* implementation briefly in V-A1. Although it is the most accurate, this configuration does not run fast enough to be usable in any scenario, so we do not consider it further here.

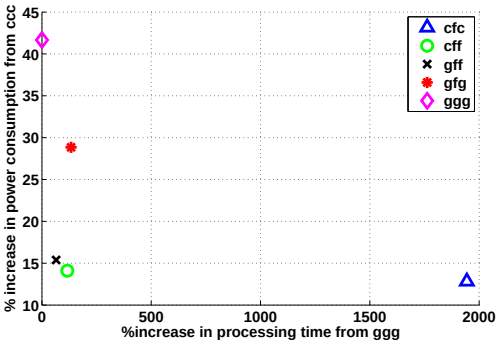
Of the configurations which are viable, *ccc* and *ggg* are compared in Figure 8. These provide suitable extremes (slowest, lowest-power, and fastest, highest-power) to compare other heterogeneous versions to.

Figure 9a shows the increase in power consumption above the *ccc* baseline of 156W against processing time. Here, all versions except *gfg* are Pareto-efficient when compared with these two characteristics. Thus, *ggg* provides a fast, high-power version, *cfc* provides a lower-power, much slower alternative, and the heterogeneous processing options occupy the intermediate region. The remaining graphs here are informative when choosing a particular configuration based on relative priorities; Figure 9b shows that, to get a 25% decrease in power, we must accept a 50-100% increase in processing time. This is expressed more accurately and intuitively in Figure 11, which uses *ggg* as a baseline. Here, an 18% reduction in power consumption from the maximum is obtained for a 50% increase in processing time (by choosing *gff*). Any desired further saving in power means that detection speed slows down considerably; from *gff*, saving an extra 2% in power (by switching to *cfc*) incurs a 20× speed penalty.

We perform a similar analysis for accuracy: Figure 10a gives

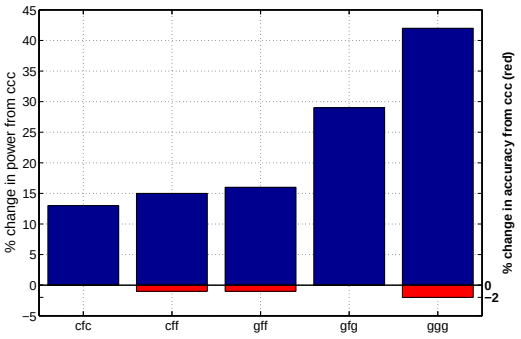


(a) Power consumption (W) above *ccc* baseline against processing time for accelerated implementations

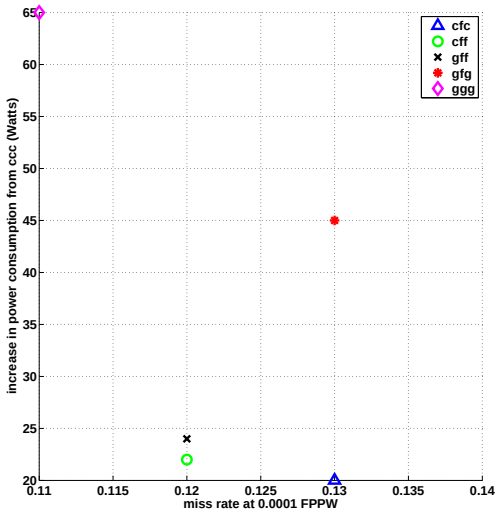


(b) Relative increase in power consumption above *ccc* vs. relative increase in time from *ggg*

Fig. 9. Power consumption and time at $n = 13$ scales



(a) Relative change in power consumption (blue) and accuracy (red) from *ccc*. A larger red bar indicates increased accuracy.



(b) Change in power consumption above *ccc* version vs. miss rate

Fig. 10. Power consumption and accuracy at $n = 13$ scales

the power vs accuracy tradeoffs. Here, a 1% improvement in miss rate at 10^{-4} FPPW requires 15% more power (using *cff*) while a 2% gain requires 40% more power, or (as shown in Figure 10b) an extra 65W (using 156W as the baseline).

From this analysis, the advantages of a heterogeneous system become apparent: in situations where speed and power consumption are both desired, and power consumption is the most important constraint, *gff* provides a suitable compromise. Note that from Table IV, *gff* requires less power than a GPU-accelerated system when no FPGA is present. As shown by Figures 9 and 10, *gfg* is always worse than other configurations in all scenarios, showing the effect of increased communications delays between processors.

In contrast to the large differences in runtime and power consumption, the difference in accuracy is relatively minor (Figure 7). As described in [9], the choice of pedestrian detection algorithm appears to have a much larger effect on accuracy than a variety of implementations. On a mobile device, high accuracy is arguably less of a priority than fast detections or low power consumption too.

When running this system, we could switch execution from one version to another in successive frames; thus, configuration selection can be done at runtime, to take account of changing circumstances in the scenarios in §I.

This example scenario becomes more informative if we expand our application to include hypothesis generation via motion or hotspot detection from an IR camera; in this mode the system would be used to confirm pedestrian detections at a certain pixel height rather than exhaustive evaluation over n scales. In the analysis above, we used $n = 13$ scales. At 13 scales per octave, this will detect pedestrians up to around 240 pixels in height. For detections closer to the camera than this, we aim to use motion-cued detection on a smaller region rather than evaluation of all possible scales across the whole

image. In any case, for low n_{scales}, the tradeoffs change: *cff* provides the fastest, lowest-power implementation, while at $n = 3$, *gff* is slightly faster and more accurate.

1) RBF Classification Tradeoffs: As discussed in §III-E, we calculated the requirements for radial basis function classifier evaluation on FPGA. Examples exist in the literature, such as Irick et al. [32] which works on small windows, or Cadambi et al. [33] which classifies at 14GMACS. However, using all the optimisations detailed in [33], and using all the multiplier resources on our FPGA, we would still take around $700ms$ to classify at single scale, and several seconds to classify a frame at multiple scales. This is due to the large n_{sv} (around 4000) generated during RBF classifier training (see §III-E) and our large descriptor length (3780). The results for *gfg-kernel* and *ggg-kernel* in Table III confirm this. Although we provide an analysis of tradeoffs of other versions, an implementation which takes this long to run is not useful in a realistic scenario. Unlike Bauer et al. [22], we do not generate hypotheses to reduce the number of windows and must evaluate all sliding windows over all scales.

B. System Evaluation

We compared a heterogeneous system against the more common model of a system with a processor and a single type of accelerator. The tradeoffs between different versions were then evaluated, in §V-A. As shown in Figure 9a, a heterogeneous system (*gff*) provides a better tradeoff between the fast, high-power all-GPU version and the slower, lower-power all-CPU version. As shown in Figure 6, this is also more accurate than existing FPGA implementations of HOG. This system would better suit a task where several algorithms (e.g. detections, segmentation etc.) can be run in parallel and the result combined at the end. In our current setup, transferring data between two accelerators over PCIe requires a transfer back to the host first; future GPU architectures allow direct FPGA-GPU transfer which would resolve this. This system is also extensible to other image processing applications such as detection of vehicle orientation or stop signs, both of which can be done using extraction of features based on local histograms followed by SVM classification. In this case the tradeoffs will also be similar.

C. Future work

Having provided a comprehensive analysis of the differences between each configuration, our next steps are to demonstrate dynamic, programmatic platform selection at runtime by switching between processing platforms in one of our example scenarios, using knowledge of the characteristics of each version to respond to changing requirements and energy constraints.

This allows extension of the tradeoff analysis to a more complex set of tasks (including object detection using multi-modal sensors). Implementing this detector in an embedded system consisting of a low-power processor and GPU with the FPGA could be done relatively easily, and would greatly reduce the idle power consumption of this system. This would be more effective for a situational awareness task when paired with the right selection of algorithms: detection performance could be improved further by better algorithm selection. The limiting factor for performance of the FPGA version is evaluation at multiple scales per octave. This implementation could be expanded to a detector with either minimal [11] or no [34] image scaling requirements which instead relies on model scaling.

ACKNOWLEDGMENT

This work was supported by funding from EPSRC through the Institute for System Level Integration, and Thales Optronics.

REFERENCES

[1] M. Komorkiewicz, M. Kluczewski, and M. Gorgon, "Floating point HOG implementation for real-time multiple object detection," in *22nd International Conference on Field Programmable Logic and Applications (FPL)*, Aug. 2012, pp. 711–714.

[2] O. Mateo Lozano and K. Otsuka, "Real-time Visual Tracker by Stream Processing," *Journal of Signal Processing Systems*, vol. 57, no. 2, pp. 285–295, Jul. 2008.

[3] J. Chase, B. Nelson, J. Bodily, Z. Wei, and D.-J. Lee, "Real-Time Optical Flow Calculations on FPGA and GPU Architectures: A Comparison Study," in *16th International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2008, pp. 173–182.

[4] J. Letham, N. M. Robertson, and B. Connor, "Contextual smoothing of image segmentation," in *International Conference on Computer Vision and Pattern Recognition Workshops, CVPRW 2010*, San Francisco, 2010, pp. 7–12.

[5] B. Connor, I. Carrie, J. Letham, and N. Robertson, "Scene understanding and task optimisation using multimodal imaging sensors and context: a real-time implementation," *SPIE Infrared Technology and Applications*, vol. 8012, pp. A1–9, 2011.

[6] P. Newman, G. Sibley, M. Smith, M. Cummins, A. Harrison, C. Mei, I. Posner, R. Shade, D. Schroeter, L. Murphy, W. Churchill, D. Cole, and I. Reid, "Navigating, Recognizing and Describing Urban Spaces With Vision and Lasers," *The International Journal of Robotics Research*, vol. 28, no. 11-12, pp. 1406–1433, Jul. 2009.

[7] N. Bellotto and H. Hu, "Multisensor-based human detection and tracking for mobile service robots." *IEEE transactions on systems, man, and cybernetics. Part B*, vol. 39, no. 1, pp. 167–81, Feb. 2009.

[8] D. Geronimo, A. M. Lopez, A. D. Sappa, and T. Graf, "Survey of pedestrian detection for advanced driver assistance systems." *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 7, pp. 1239–58, Jul. 2010.

[9] P. Dollar, C. Wojek, B. Schiele, and P. Perona, "Pedestrian Detection: An Evaluation of the State of the Art," *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 4, pp. 743–762, Jul. 2011.

[10] P. Dollár, R. Appel, and W. Kienzle, "Crosstalk Cascades for Frame-Rate Pedestrian Detection," in *ECCV 2012*, 2012, pp. 1–14.

[11] P. Dollar, S. Belongie, and P. Perona, "The Fastest Pedestrian Detector in the West," in *Procedings of the British Machine Vision Conference, BMVC 2010*. British Machine Vision Association, 2010, pp. 68.1–68.11.

[12] Willow Garage, "OpenCV v.2.4," 2011.

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

[14] B. Cope, P. Y. Cheung, W. Luk, and L. Howes, "Performance Comparison of Graphics Processors to Reconfigurable Logic: A Case Study," *IEEE Transactions on Computers*, vol. 59, no. 4, pp. 433–448, Apr. 2010.

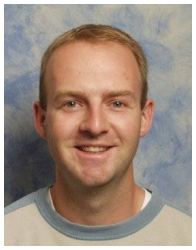
[15] M. Zuluaga and N. Topham, "Resource Sharing in Custom Instruction Set Extensions," in *2008 Symposium on Application Specific Processors*. IEEE, 2008, pp. 7–13.

[16] N. Robertson and J. Letham, "Contextual Person Detection in Outdoor Scenes," in *Proc. European Conference on Signal Processing (EUSIPCO 2012)*, 2012.

[17] L. Wolf and S. Bileschi, "A Critical View of Context," *International Journal of Computer Vision*, vol. 69, no. 2, pp. 251–261, Apr. 2006.

[18] S. Matzka, Y. Petillot, and A. Wallace, "Efficient Resource Allocation using a Multiobjective Utility Optimisation Method," in *Workshop on Multi-camera and Multi-modal Sensor Fusion Algorithms and Applications-M2SFA2*, 2008.
[19] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-e. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. V. Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney, "Stanley : The Robot that Won the DARPA Grand Challenge," *Journal of Field Robotics*, vol. 23, no. April, pp. 661–692, 2006.
[20] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, CVPR 2005. IEEE Computer Society Conference on*. IEEE Computer Society, 2005, pp. 886– 893.
[21] V. Prisacariu and I. Reid, "fastHOG-a real-time GPU implementation of HOG," Department of Engineering Science, Oxford University, Tech. Rep. 2310, 2009.
[22] S. Bauer and S. Kohler, "FPGA-GPU architecture for kernel SVM pedestrian detection," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Conference on*. IEEE, 2010, pp. 61–68.
[23] R. Kadota and H. Sugano, "Hardware architecture for HOG feature extraction," in *Intelligent Information Hiding and Multimedia Signal Processing, 2009. IIH-MSP'09. Fifth International Conference on*. IEEE, 2009, pp. 1330–1333.
[24] S. Martelli, D. Tosato, M. Cristani, and V. Murino, "FPGA-based pedestrian detection using array of covariance features," *Distributed Smart Cameras (ICDSC), 2011 Fifth ACM/IEEE International Conference on*, pp. 1–6, 2011.
[25] M. Hiromoto and R. Miyamoto, "Hardware architecture for high-accuracy real-time pedestrian detection with CoHOG features," in *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*. IEEE, Sep. 2009, pp. 894—-899.
[26] P. E. P. Rybski, D. Huber, D. D. Morris, and R. Hoffman, "Visual classification of coarse vehicle orientation using Histogram of Oriented Gradients features," in *2010 IEEE Intelligent Vehicles Symposium*. IEEE, Jun. 2010, pp. 921–928.
[27] T. P. Cao and G. Deng, "Real-Time Vision-Based Stop Sign Detection System on FPGA," in *Digital Image Computing: Techniques and Applications*. IEEE, 2008, pp. 465–471.
[28] H. Quinn, M. Leeser, and L. Smith King, "Dynamo: a runtime partitioning system for FPGA-based HW/SW image processing systems," *Journal of Real-Time Image Processing*, vol. 2, no. 4, pp. 179–190, 2007.
[29] T. Wilson, M. Glatz, and M. Hodlmoser, "Pedestrian detection implemented on a fixed-point parallel architecture," in *Consumer Electronics, 2009. ISCE'09. IEEE 13th International Symposium on*. IEEE, 2009.
[30] S. Bauer, U. Brunsmann, and S. Schlotterbeck-macht, "FPGA Implementation of a HOG-based Pedestrian Recognition System," in *MPC Workshop Karlsruhe*, no. July, 2009.
[31] NVidia Corporation, "Developing a Linux kernel module using RDMA for GPUDirect," 2012.
[32] K. Irick, M. DeBole, V. Narayanan, and A. Gayasen, "A Hardware Efficient Support Vector Machine Architecture for FPGA," in *16th International Symposium on Field-Programmable Custom Computing Machines*. IEEE, Apr. 2008, pp. 304–305.
[33] S. Cadambi, I. Durdanovic, V. Jakkula, M. Sankaradass, E. Cosatto, S. Chakradhar, and H. P. Graf, "A Massively Parallel FPGA-Based Coprocessor for Support Vector Machines," in *17th IEEE Symposium on Field Programmable Custom Computing Machines*. IEEE, 2009, pp. 115–122.
[34] R. Benenson and M. Mathias, "Pedestrian detection at 100 frames per second," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, 2012, pp. 2903–2910.

Calum Blair Calum Blair is an Engineering Doctorate (EngD) student with the Visionlab at Heriot-Watt University, Edinburgh, and Thales Optronics. His doctorate studies focus on the real-time acceleration of image processing and scene analysis algorithms across various hardware platforms. He received his M.Eng. from the University of Glasgow in 2009.



Neil M. Robertson Dr Neil Robertson is principal investigator of the Vision Lab at Heriot-Watt University and an Honorary Fellow of the School of Engineering at the University of Edinburgh. He works on projects in collaborative robotics, human behaviour recognition, multi-modal registration and sensor fusion with his research team (http://visionlab.eps.hw.ac.uk). From 2000-07 Dr Robertson worked in the UK Scientific Civil Service with DERA, and then QinetiQ ltd. He held a prestigious 1851 Royal Commission Fellowship at Oxford University (2003-06) in the Robotics Research Group. He received his D.Phil. from Oxford in 2006 and M.Sci. from Glasgow University in 2000.

Danny Hume Danny Hume has a B.Eng.(Hons) in Electrical Engineering and is manager of the Systems Discipline at Thales Optronics, a defence company specialising in optics and imaging systems and systems integration. He is a member of the IET.