A Characterization of Visual Feature Recognition

Binu Mathew, Al Davis, Robert Evans
$\{$*mbinu* $|$ *ald* $|$ *revans*$\}$*@cs.utah.edu*

UUCS-03-014

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

September 3, 2003

Abstract

Natural human interfaces are a key to realizing the dream of ubiquitous computing. This implies that embedded systems must be capable of sophisticated perception tasks. This paper analyzes the nature of a visual feature recognition workload. Visual feature recognition is a key component of a number of important applications, e.g. gesture based interfaces, lip tracking to augment speech recognition, smart cameras, automated surveillance systems, robotic vision, etc. Given the power sensitive nature of the embedded space and the natural conflict between low-power and high-performance implementations, a precise understanding of these algorithms is an important step developing efficient visual feature recognition applications for the embedded space. In particular, this work analyzes the performance characteristics of flesh toning, face detection and face recognition codes based on well known algorithms. We also show how the problem can be decomposed into a pipeline of filters that have efficient implementations as stream processors.

1 Introduction

The focus on embedded computing has both diversified and intensified in recent years as the focus on mobile computing, ubiquitous computing, and traditional embedded applications have begun to converge. A side effect of this intensity is the desire to support sophisticated applications such as speech recognition, visual feature recognition, and secure wireless networking in a mobile, battery-powered platform. Unfortunately these applications are currently intractable for the embedded space. Running these applications on a low-power embedded processor cannot keep up with the inherent real-time processing requirements of speech recognition for example. The problem is that low-power processors do not have sufficient compute power. Using mainstream high-performance microprocessors comes close to meeting the performance requirements but the energy requirements are not commensurate with mobile and embedded processing domains. The first step in developing new architectures and systems which can adequately support these applications is a precise understanding of the algorithms.

Our focus has been on perception algorithms which form the basis for natural human interfaces to the embedded computing space. This is motivated by a belief that natural human interfaces are essential to realizing the dream of ubiquitous computing. In earlier work we have studied speech recognition [8] and the results of that study have allowed us to create new architectures which support real-time, speaker independent, large vocabulary speech to text applications at a power level that is commensurate with embedded space energy budgets. In this paper, we report on a similar study of visual feature recognition systems.

Visual feature recognition systems vary significantly based on the type of feature that is being recognized. Relatively simple recognizers are regularly employed in industrial visual inspection systems. On the other hand, human face recognition is an extremely complex task given the huge possibility space of facial features and skin tones. Facial recognition systems clearly have utility in security and surveillance domains, and other visual recognizers play key roles in gesture interfaces, lip reading to support speech recognition, and robotics. Our interest in face recognition however is primarily motivated by the difficulty of the problem which cannot be currently supported by embedded systems. Furthermore the structure of our face recognizer appears to be easily adapted to address other visual feature recognition tasks. The main differences for these other tasks is a different training regimen and different frame rate requirements. For example, the Rowley method of face detection described in this paper has been applied to license plate detection [10].

The particular application studied here can be viewed as a pipeline of 3 major functional components. A flesh tone detector is used to isolate areas of a frame where a face is likely to be present. The next stage is a face detector which determines whether there is a face present or not. The final phase is a face recognizer. Each of these components is based on well known algorithms which have been adapted to fit into our framework. Some algorithmic optimization and restructuring has been done to suit our purposes but the basic approach has been developed by other researchers.

Interestingly the face recognition system when viewed from a structural perspective comprises a series of increasingly discriminating filters. Early stages of the sequence must inherently filter the entire image. As the process proceeds downstream, each stage needs to examine less image data since previous stages have eliminated certain area from the probable candidate list. The result is an interesting balance of simple algorithms which analyze lots of data early in the sequence and more sophisticated algorithms which only need to analyze limited amounts of data late in the process. The result is a structure which is amenable for implementation as an embedded system.

The following sections provide a conceptual description of the recognition process, characterize our test harness, describe the performance and power characteristics of the recognizer both by phase and for the whole application, describe optimization options, and concludes.

2 Overview of Visual Feature Recognition

Figure 1 shows the major steps in face recognition. The input is a low-resolution video stream such as 320x200 pixels at 10 frames per second. The stream is processed one frame at a time and sufficient state is maintained to perform history sensitive tasks like motion tracking. The process is essentially a pipeline of filters which reduce the data and attach attributes to frames for the use of down stream components. Typically each filter is invoked at the frame rate. This underlines the soft real time nature of this application. Additional data is required since filters may access large databases or internal tables. These additional data streams add to the aggregate bandwidth requirement of the system. The periodic nature of the application domain often makes it possible to easily estimate the worst case requirements.

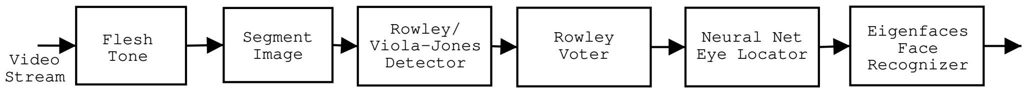

Figure 1: Algorithmic stages of a face recognizer

Object recognition typically proceeds in two steps: object detection and the actual object identification. Most approaches to object identification require a clearly demarked area, normalized to a particular size, and the location of key features. Object detectors find the area where the desired feature is likely to reside, scale the area to meet the normalization requirement, and then creates a location and boundary description for that area. False positives and negatives occur but the algorithms try and minimize their occurrence.

Object detectors also often work at a fixed scale. The detector is swept across the image recording all positions at which a detection was reported. The image is then sub-sampled or scaled down by a small factor (typically 0.8) and the process is repeated until the frame is below the size of the detector. A decision procedure is then applied to all the predicted hits to decide which ones are the most likely. Detectors often have much lower compute cost per sub-window than their corresponding identifying routines. Since they are swept across the entire image, a significant portion of the applications execution time might be spent in the detector. In contrast, even though identifying filters are more compute intensive, they are applied only to the high probability regions of the frame, so their contribution to the overall execution time might be low. Though object detectors are less compute intensive, they are much more difficult to design due to their generality. For example a face identifier chooses from one of N known faces, but a face detector has to distinguish between the infinite sets of faces and non-faces.

Since detection is time consuming, it is common to structure an object detector as a cascade of filters with cheaper heuristics upstream identifying potential regions for more expensive heuristics downstream. An extreme case of this is the Viola/Jones method which trains a sequence of about 200 increasingly discriminate filters [14]. A more common approach when dealing with faces and gestures is to identify the flesh colored regions of an image and apply a more sophisticated detector to those regions.

The identifier receives candidate regions from the detector along with other information like probability, scale and feature locations. It typically identifies key features of interest, say edges or connected components and employs some sort of distance metric from known references to provide a positive identification. In our face recognizer, the first level of detection is provided by flesh toning which is followed by an image segmenting algorithm. These are followed in turn by aa more complex detector, voting for high probability regions, an eye locator and finally a face identifier.

2.1 Flesh Toning

Flesh toning identifies flesh colored pixels in an image. The commonly used RGB color space is not well suited for flesh toning because skin color occupies a wide range in primary color space. Variations due to lighting and ethnicity are hard to deal with and skin-like colors on walls and clothing are harder to discriminate. However, skin colors are tightly clustered in color spaces like HSV and YCbCr. Flesh toning can be done by converting pixels from sample images into the chosen color space and making a scatter plot with two colors, one for flesh pixels and one for non-flesh pixels. A boundary is then drawn around flesh tone clusters. This boundary is then approximated by curves which can be described by simple geometric equations. In the image under test, any pixel that lies inside this new approximated but easily described boundary is considered to be a flesh pixel.

The base algorithm involves transforming the RGB color space into the NCC (Normalized Color Coordinates) space using the simple equation $r = R/(R + G + B)$, $g = G/(R + G + B)$. In this space flesh pixels occupy a space bounded by two parabolas and maximum and minimum x-axis values. Applying two inequalities of the form $ax^2 + bx + c$ to the color coordinates will predict if the pixel is flesh colored or not [12]. While this algorithm is simple and achieves good discrimination, we observed that it tends to classify certain shades of blue found in clothing as a skin color. A second algorithm is then employed to transform the RGB value of a pixel to an HSV (Hue, Saturation, Value/Luminance) value. In the HSV space, flesh color is tightly clustered allowing the use of four simple inequalities for flesh tone [3]. In practice we have observed that this algorithm generates too many false positives. However, when we employ the consensus of the two color space algorithms the results are very good. The output of this phase is a bit mask of the same size as the image where bits are set if the corresponding pixel is flesh colored.

2.2 Segmentation

Segmentation is the process of clumping together individual pixels into regions where an object might be found. A common approach is to do a connected component analysis which typically forms irregular regions. Since the the Viola and Rowley algorithms we use for face detection need rectangular regions, instead of connected component analysis we use a simple algorithm to cut apart the flesh tone bit mask into rectangles [14, 10].

First two operators from mathematical morphology are applied to the bit mask: a 3x3 erosion operator followed by a 4x4 dilation operator. This has the effect of cutting away small connections and regions which are likely to be false positives and then smoothing the bit mask by filling in any small holes in the middle of an otherwise acceptable sized region. A logical OR of all the rows in the image is then performed to make a single row. This step is called vertical separation. Runs of "1" values in the single row represent vertical stripes of the image that contain objects of interest. Runs of "0" values represent vertical stripes that may be thrown away. For each vertical stripe, we logically "OR" the columns to create a single column. This is called horizontal separation. Runs of “1” represent the region of interest. This algorithm can be recursively applied to isolate the rectangular regions of interest. In the actual implementation, the horizontal separation steps for all the vertical stripes are done together in an interleaved manner. This has the effect of converting the column walk across the bitmap into a row walk giving better cache performance. Recursion is stopped after two levels since this has empirically provided adequate results. The flesh tone bitmap is discarded at this stage. The output of this stage is a list of coordinates of the top left and bottom right corners of rectangular regions of interest and a gray scale version of the image.

2.3 Rowley Face Detector

Henry Rowley’s neural net based face detector is well known as a pioneer in this field [10]. Its implementation was provided to us by the Robotics Institute at CMU. This detector is designed to determine if a 20x20 pixel image contains a face or not. Face detection is done by sweeping the detector over the image and computing the decision at each pixel location. Then the image is scaled and reduced in size by a factor of 0.8 and the procedure is repeated. The resulting series of images and detection locations is called an image pyramid. In the case of real faces, a detection will be reported at several nearby pixel locations at one scale and at corresponding locations in nearby scales. False positives do not usually happen with this regularity. Hence a voting algorithm can be applied to the image pyramid to decide the site of any true detections.

In each window the detector first applies a correction for varying lighting conditions followed by histogram equalization to expand the range of intensity values. The preprocessed window is then applied to a multi-layer neural network where the input layer has retinal connections to the image window. This is followed by a hidden layer comprised of three classes of units. Four units look at 10x10 sub-windows, 16 units look at 5x5 sub-windows and 6 units look at overlapping 20x5 horizontal stripes. The final output of the network indicates if the 20x20 window contains a face or not.

The *voting algorithm* (our terminology) notes the location and scale of each detection in an image pyramid. The next step called *spreading* replaces each location in the pyramid with the count of the number of detections in a neighborhood. The neighborhood of a location extends an equal number of pixels along the position and scale axes. The values are then thresholded and the centroids of all remaining locations are found. Centroids are examined in descending order of the number of detections per centroid and other centroids that represent a face overlapping the current face are eliminated. The remaining centroids represent the location of faces found in the image. To further reduce false positives, multiple neural nets each trained separately may be applied to the image and their consensus can represent a more accurate detection.

2.4 Viola and Jones' Detector

Viola and Jones present a new and radically faster approach to face detection based on the AdaBoost algorithm from machine learning [14]. They claim a 15x speedup over the Rowley detector for their implementation without using flesh toning. Since their source code is proprietary, we re-implemented their algorithm based on example code obtained from the University of British Columbia. Our re-implementation uses flesh-toning for both the Rowley and the Viola/Jones detectors and the algorithms are close in speed due to factors mentioned later. To understand the Viola/Jones detector we first need to explain the concept of *boosting*.

A random guess to a yes or no question stands the chance of being correct 50% of the time. If a heuristic can improve the odds by a very small amount then it is called a *weak learner*. It is possible to generate weak learners for several tasks in a semiautomated manner by enumerating a huge set of heuristics generated on the basis of combinations of simple rules and evaluating their performance on a set of samples. A heuristic that can improve the odds of the guess by a significant amount is called a *strong learner*. Boosting is a method of combining several weak learners to generate a strong learner. AdaBoost is a well known algorithm to generate strong learners from weak learners, while providing statistical bounds on the training and generalization error of the algorithm [11].

The weak learners in the Viola/Jones algorithm are based on *features* of three kinds. A two rectangle feature is the difference between the sum of the values of two adjacent rectangular windows. A three rectangle feature considers three adjacent rectangles and computes the difference between sum of the pixels in the extreme rectangles and the sum of the pixels in the middle rectangle. A four rectangle feature considers a 2x2 set of rectangles and computes the difference between sum of pixels in the rectangles that constitute the main and off diagonals. For a 24x14 sub-window there could be more than 180,000 such features. The task of the AdaBoost algorithm is to pick a few hundred features and assign weights to each using a set of training images. Face detection is reduced to computing the weighted sum of the chosen rectangle-features and applying a threshold. As in the case of the Rowley algorithm a 24x24 detector is swept over every pixel location in the image and the image is rescaled. We apply Rowley's voting algorithm to decide the final detection locations.

The original slow approach described in the Viola/Jones paper uses 200 features. They then go on to describe a faster approach where they cascade many such detectors with more complex detectors following simpler ones. A window is passed to a detector only if it was not rejected by the preceding detector. Since training this cascade is a laborious process, we model the workload characteristics of this algorithm with a single 100 feature detector.

2.5 Eigen Faces

Eigenfaces is a well known Principle Component Analysis (PCA) based face recognition algorithm developed by researchers at MIT [13]. In this paper, we use a re-implementation of the Eigenfaces algorithm from researchers at Colorado State University [5]. Face images are projected onto a feature space called *face space* defined by the eigen vectors of a set of faces. This captures the variation between the set of faces without emphasis on any one facial region like the eyes or nose. The mathematical treatment of Eigenfaces is too involved to discuss here. The approach works by computing and storing the face space corresponding to each face in a training set. A test image is projected on to each saved eigenface and a set of weights is computed based on closeness to the known eigenfaces. The weights are then used to label the test image as one of the known persons or an unknown one. In our evaluation the eigenfaces database contains 10 personalities from television.

3 Characterization

In this section we provide a detailed characterization of visual feature recognition by native execution and profiling using processor performance counters as well as via simulation. The native execution results were obtained using SGI SpeedShop on a 666 MHz R14K processor. Simulation studies are based on MLRSIM, an out of order processor simulator derived from the Rice University RSIM simulator. It is detailed enough to run a derivative of the Net BSD operating system and can run SPARC binaries compiled for Sun OS without any modification. A multi-GHz processor is required to operate this application in real time. Parameters like L1 cache hit time, memory access time, floating point latencies etc. were measured on a 1.7 GHz AMD Athlon processor using the lmbench hardware performance analysis benchmark [9]. Numbers that could not be directly measured were obtained from vendor micro-architecture references. MLRSIM was configured to reflect these parameters. Unless mentioned otherwise, the remainder of this paper uses the default configuration.

Default Configuration: SPARC V8 ISA, 2 GHz clock frequency, 16KB 2 way associative L1 I and D cache with 2 cycle latency, 2MB L2 Cache, 2 way associative, 20 cycle latency, max 4 integer + 4 floating point issue, max 4 graduations per cycle, 600 MHz, 64 bit DRAM interface.

Embedded Configuration: This closely models an Intel Xscale $StrongARM$ development system with the exception that the processor has a floating point unit. SPARC V8 ISA, 400 MHz clock frequency, 32 KB 32 way associative L1 I and D cache with 1 cycle latency, 1 ALU, 1 FPU, single issue, 100 MHz 32 bit DRAM interface. Though the XScale does not have an L2 cache, since MLRSIM cannot be configured without an L2 cache, this configuration has a 64KB *inclusive* L2 cache. Since the cache is inclusive and the same size as the sum of the L1 caches, this configuration behaves similar to a machine with no L2 cache.

We discuss characteristics of the application in 5 configurations: a) full pipeline using the Rowley face detector, b) full pipeline using the Viola/Jones face detector, c) only the Rowley face detector with flesh toning and image segmentation, d) only the Viola/Jones face detector with flesh toning and image segmentation, e) only the Eigenfaces recognizer. The last three configurations are important from an energy savings perspective since running the individual algorithms on separate low frequency processors or hardware accelerators, can lead to significant energy savings.

Figures 2 and 3 show the relative execution times of each algorithm using the Rowley detector and the Viola/Jones detector. Figures 4 and 5 show the L1 Dcache miss rate and the L2 cache hit rates for all 5 application configurations. Since the caches are inclusive, the L2 hit rate is defined as the L1 misses that hit in the L2 cache divided by the total number of accesses made by the application. Since this application achieves 99.8% ICache hit rate with a 16KB ICache, no other ICache configurations were studied. Figure 6 shows IPC for a variety of execution unit configurations and Figure 7 shows the run times normalized to real-time. Here, 1.0 represents minimum real time performance corresponding to 5 frames per second. For example, in Figure 7 in the 1 ALU + 1 FPU configuration, the Rowley algorithm is 1.13 times slower than real-time while the Eigenfaces algorithm processes 5 frames in 0.69 seconds.

For the entire application there is consistently greater than 92% L1 cache hit rate for D-caches of 16KB and above. This indicates that the streaming pipelined model we use for composing the algorithms is a good fit for the problem. Each 320x200 pixel color image is 187.5 KBytes long and the corresponding gray scale versions are about 64 KBytes. The images clearly will not fit in the L1 cache. The explanation is that the color image is accessed in streaming mode, i.e. each pixel is touched exactly once for flesh toning. Image segmentation works on the flesh tone bitmap (approximately 64KB) making at most two passes over it. Since these accesses touch at most two image rows at a time, good cache utilization is insured. Subsequently, only small windows into the image are used. Since

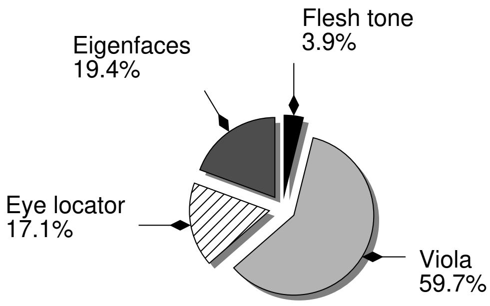

Figure 2: Execution time break down of Viola/Jones detector based face recognizer

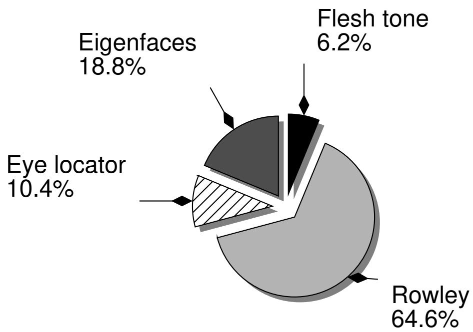

Figure 3: Execution time break down of Rowley detector based face recognizer

objects in these images are typically smaller than 50x50 pixels, each object is only about 2.5KB in size. The downstream algorithms make several passes over each object, but only a small part of each object needs to be cache resident at each time. For example, the integral image computation in the Viola/Jones algorithm is based on a recurrence that involves two adjacent image rows and an additional row for intermediate storage and has an L1 cache footprint of about 4.4KB. The Rowley algorithm touches at most 20 rows of the object at the same time. However, as it sweeps across the image left to right and top to bottom only a 20x20 pixel window needs to be cache resident at a time. Since it shifts its position one pixel at a time, a 19x19 region of this window will be reused by the next iteration contributing to high L1 cache hit rate. A similar pattern occurs in the later phase of the Viola/Jones algorithm on a 24x24 region. The Eigenfaces algorithm uses a projected image of the object to be recognized as well as basis, mean and projected image matrices corresponding to each reference object. The target object is reused while it is compared

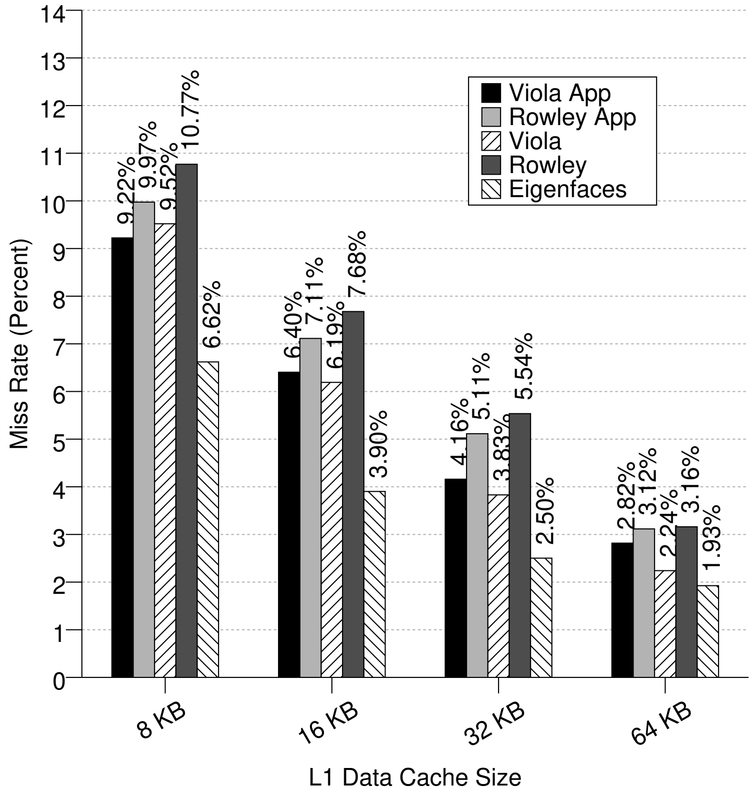

Figure 4: L1 Dcache miss rate

against each candidate. Each candidate however is accessed only once per target object.

The objects and their attributes from each stage are typically touched again by the next stage. The auxiliary information used by the algorithms is somewhat small. Both detector algorithms use fixed size data structures. The worst case is the Viola/Jones algorithm which needs a weight and a type for each feature corresponding to 100 * 2 * 4 = 800 bytes of L1 cache. The data set for the Eigenfaces algorithm on the other hand is linear in the number of the reference faces. But since these could potentially be streamed into the L1 Dcache once per target object (or once per frame) its foot print is small. Only the projected target object and a small part of the basis/mean/projected reference images need to be resident in the L1 Dcache. From Figure 5 it is seen that the L2 cache is largely ineffective since it is accessed infrequently due to the low L1 miss rate.

From a cache footprint perspective, both the detector algorithms and the entire application appear to be a good match for embedded processors with limited cache resources. Since

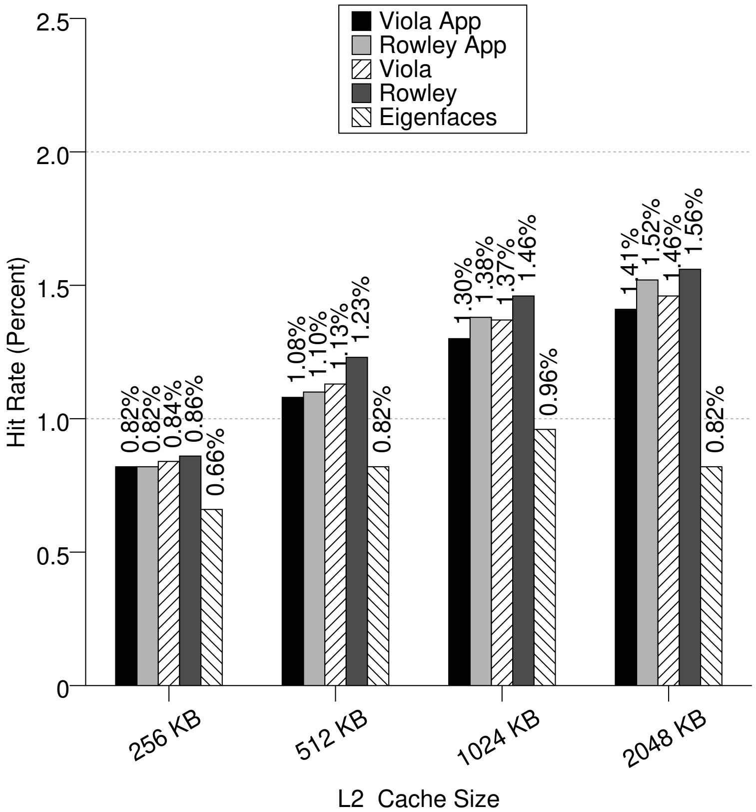

Figure 5: L2 Cache hit rate

images are accessed left to right, multiple rows at a time, sequential prefetch (or strided prefetch) would hide memory access latencies even when the L1 Dcache is small. However, quite a different view unfolds on examining the IPC and speedup graphs. It is seen that embedded processors are inadequate to handle the work load in real time. In this case the execution bandwidth is the culprit, not the memory system. The power budgets required for such performance are beyond what is available on normal low power embedded platforms. Thermal dissipation is a problem even on high performance processors and energy saving solutions are important for real time workloads like visual feature recognition.

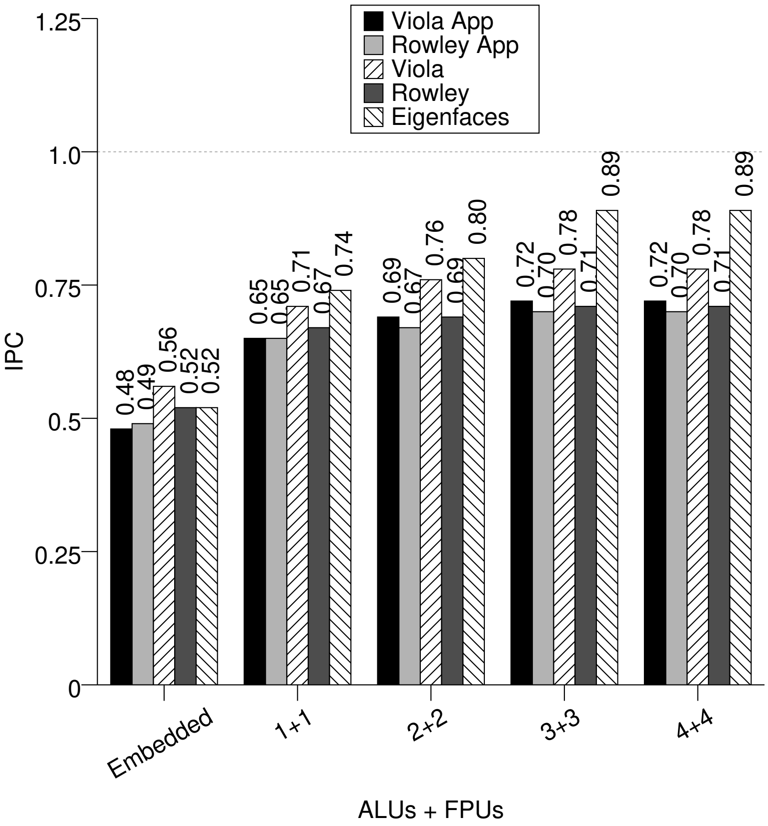

Figure 6: IPC

4 Software Optimizations

The integral image computation in the Viola/Jones algorithm is an excellent example of the role of algorithm level optimizations for this domain. The integral image value at pixel location (x,y) in an image is defined as the sum of all pixels to the left and above the pixel (x,y). This is computationally prohibitive. By expressing the same relationship as a pair of recurrences, it is possible to compute the integral image with just one pass over the image. Similarly, our initial implementation computed the standard deviation of all pixels within a 24x24 pixel window starting at each pixel location within the region of an image where flesh toning predicted a target. This was seen to occupy between 10-15% of the compute time of the whole application. By defining a set of recurrences for the mean and mean square for 24x24 sub windows over a wider NxM region, we were able to compute the standard deviations in one pass over the image thereby reducing the execution time of this

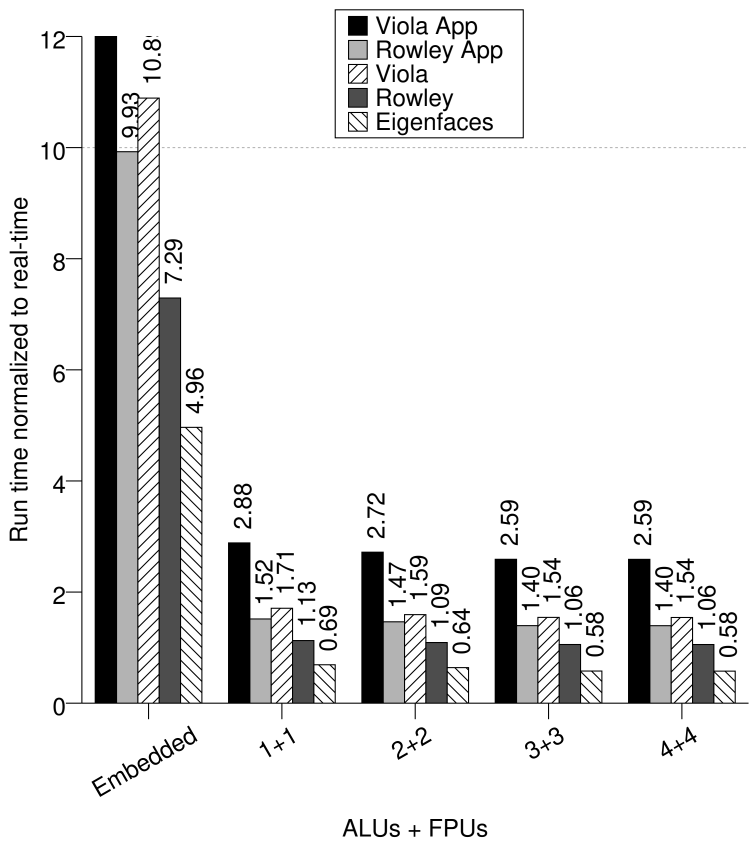

Figure 7: Speedup or Slow down over real time

component to less than 1%. However, such transformations require a lot of attention from the programmer and insight into the algorithm. Executing kernels that operate on a MxM sub-window of a larger NxN image and then sliding the window is a recurring theme in image processing. Reordering the computation so that data may be streamed through a set of execution units and computed in the minimum number of passes while observing limits on the amount of intermediate results has wide applicability. Compiler based tools that can take M, N, a kernel, and the size of the intermediate storage available and effect the transformation automatically have an opportunity to enhance performance and reduce resource utilization significantly.

5 Hardware Optimizations

As seen in Figures 6 and 7, wide issue clearly helps performance. But wide issue comes at the cost of increased hardware complexity and could potentially limit the clock frequency as well as exceed a limited energy budget. This application is embarrassingly parallel in most sections due to the intrinsic data parallelism in the pixel processing. One way of achieving good performance at low power is to use a cluster of simple processors or function units operating in parallel with a very small quantity of SRAM for local storage and no cache. Significant power savings is possible using this approach. For example consider the Rowley filter which is approximately real time on a 2 GHz processor with 2 ALUs and 2FPUs as seen in Figure 7. The detector workload can be done in parallel on 5 simple streaming units and an embedded processor can execute other application stages. The entire ensemble operates at 400 MHz. Since we seek to establish just a lower bound on energy savings, assume that the streaming units and the embedded processor would each consume the same amount of power as the a Pentium 4 processor running at 400 Mhz. Using a non-intrusive current probe and digital oscilloscope on a PC motherboard modified at the board level we found that a 2.4 GHz Pentium 4 processor consumes approximately 56.5 Watts at 1.8 volts when operating on a compute intensive loop. Using CMOS delay equations we can calculate the approximate power consumption of the same processor running at 400 MHz and a correspondingly lower supply voltage [2]. Assuming that the threshold voltage for Intel's 0.13μ CMOS process could be in the range 0.5 to 1 volts and the CMOS parameter α is in the range 1.3 to 2.2, the power will be in the range 2.5 to 5 Watts. Operating six such units on the same die would consume 15 to 30 Watts. This can be considered a worst case bound on the energy savings possible. In our previous research with speech recognition systems we have observed that an order of magnitude or better improvement was possible in the energy delay product using a simple execution cluster/specialized compiler combination. We expect similar results for visual feature recognition.

6 Related Work

Perception processing which encompasses a wide range of topics like computer vision, speech recognition and gesture recognition is currently the focus of vigorous research. While it is common in the literature to see the relative merits and performance of algorithms compared, architecture level analysis of whole perception applications is extremely rare. Wang, Bhat and Prasanna discussed methods for implementing scalable computer vision algorithms on commercial parallel computers [16]. The Image Understanding Architecture, a parallel processor for real time machine vision and its software environment developed jointly by the University of Massachusetts and Hughes research is described in [18]. Wawrzynek et al described the performance of the SPERT II, a vector microprocessor based hardware accelerator for neural network algorithms [17]. Agaram, Keckler and Burger presented a detailed architecture level analysis of the CMU Sphinx II speech recognizer [1].

The next step in this research is to investigate special purpose architectures for machine vision. This approach has shown significant advantages in both performance and energy consumption for speech recognition [8]. Existing efforts on vision architectures can be partitioned into analog and digital approaches. The analog approaches have primarily been based on modeling the human neural system. This neuromorphic approach has produced a variety of efficient special purpose devices for early-vision functions [6, 7, 19]. Digital approaches have covered a broader spectrum of vision functions. Representative examples are commercial offerings by companies such as Cognex and Coreco (www.cognex.com, www.coreco.com) which provide application specific software for industrial applications such as visual inspection, security monitoring, motion detection, etc. These commercial approaches employ standard processor and memory architectures but may or may not dispense with a general operating system environment. Others have experimented with directly mapping algorithm flows onto FPGA based systems, and the utility of new highly parallel architectures such as the MIT RAW machine for vision applications [4, 15]. ASIC implementation of vision applications are rare, perhaps due to the significant diversity of computer vision algorithms and the high cost of ASIC development. Each approach offers distinct advantages but is also limited by intrinsic disadvantages. General purpose processors are slow when compared to ASIC approaches. Analog circuits are both fast and energy efficient but they lack generality and are costly to produce. FPGA based systems are a compromise between generality and efficiency. We feel that there is a significant opportunity for a much better compromise which involves a customizable compute cluster which retains most of the generality of the generic processor approach while achieving performance/energy efficiency levels close to that of special purpose ASIC approaches.

7 Conclusions

We have presented a detailed analysis of the performance characteristics of a face recognition system based on well-known algorithms. Existing face recognition systems are inadequate to support real-time operation on embedded systems. We have shown that by taking advantage of the stream oriented nature of the application it is possible to solve this problem. It is also evident that other visual feature recognizers can benefit from similar tactics since our face recognizer needs few modifications in order to recognize other visual features. This effort and our previous work on speech recognition systems leads to a strong belief that recasting perception algorithms into a stream oriented style is the key to improving performance, reducing power consumption, and supporting these sophisticated applications on embedded devices with limited cache, processing, and energy resources.

8 Acknowledgments

The authors would like to thank the Vision & Autonomous Systems Center (VASC) at the Robotics Institute of the Carnegie Mellon University and Tsuyoshi Moriyama in particular for providing us with the source code of the CMU/Rowley face detector. We would also like to thank Peter Carbonetto of the University of British Columbia for providing the example code which formed the starting point for our AdaBoost face detector.

References

[1] AGARAM, K., KECKLER, S. W., AND BURGER, D. A characterization of speech recognition on modern computer systems. In *Proceedings of the 4th IEEE Workshop on Workload Characterization* (Dec. 2001).

[2] ATHAS, W., YOUNGS, L., AND REINHART, A. Compact models for estimating microprocessor frequency and power. In *Proceedings of the 2002 international symposium on Low power electronics and design* (2002), ACM Press, pp. 313–318.

[3] BERTRAN, A., YU, H., AND SACCHETTO, P. Face detection project report. http://-ise.stanford.edu/2002projects/ee368/Project/reports/ee368group17.pdf, 2002.

[4] BONDALAPATI, K., AND PRASANNA, V. Reconfigurable computing systems. *Proceedings of the IEEE 90*, 7 (July 2002), 1201–1217.

[5] COLORADO STATE UNIVERSITY. Evaluation of face recognition algorithms. http://-www.cs.colostate.edu/evalfacerec/ , May 2003.

[6] HIGGINS, C. Multi-chip neuromorphic motion processing. In *Proceedings of the Conference on Advanced Research in VLSI, Atlanta* (March 1999).

[7] MAHOWALD, M. A., AND MEAD, C. *Analog VLSI and Neural Systems*. Addison-Wesley, 1989, ch. Silicon Retina, pp. 257–277.

[8] MATHEW, B., DAVIS, A., AND FANG, Z. A gaussian accelerator for sphinx 3. Tech. Rep. UUCS-03-002, School of Computing, University of Utah, 2003.

[9] MCVOY, L. W., AND STAELIN, C. lmbench: Portable tools for performance analysis. In *USENIX Annual Technical Conference* (1996), pp. 279–294.

[10] ROWLEY, H. A., BALUJA, S., AND KANADE, T. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence 20*, 1 (1998), 23–38.

[11] SCHAPIRE, R. E. The boosting approach to machine learning: An overview. In *In MSRI Workshop on Nonlinear Estimation and Classification* (2002).

[12] SORIANO, M., MARTINKAUPPI, B., HUOVINEN, S., AND LAAKSONEN, M. Using the skin locus to cope with changing illumination conditions in color-based face tracking. In *Proceedings of the IEEE Nordic Signal Processing Symposium* (2000), pp. 383–386.

[13] TURK, M., AND PENTLAND, A. Face recognition using Eigenfaces. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)* (June 1991), pp. 586–591.

[14] VIOLA, P., AND JONES, M. Rapid object detection using a boosted cascade of simple features. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (Dec. 2001).

[15] WAINGOLD, E., TAYLOR, M., SRIKRISHNA, D., SARKAR, V., LEE, W., LEE, V., KIM, J., FRANK, M., FINCH, P., BARUA, R., BABB, J., AMARASINGHE, S., AND AGARWAL, A. Baring it all to software: Raw machines. *IEEE Computer 30*, 9 (1997), 86–93.

[16] WANG, C.-L., BHAT, P. B., AND PRASANNA, V. K. High performance computing for vision. *Proceedings of the IEEE 84*, 7 (July 1996), 931–946.

[17] WAWRZYNEK, . J., ASANOVIC, K., KINGSBURY, B., BECK, J., JOHNSON, D., AND MORGAN, N. Spert-ii: A vector microprocessor system. *IEEE Computer 29*, 3 (March 1996), 79–86.

[18] WEEMS, C. C. The second generation image understanding architecture and beyond. In *Proceedings of Computer Architectures for Machine Perception* (Nov. 1993), pp. 276–285.

[19] WYATT, J. L., KEAST, J. C., SEIDEL, M., STANDLEY, D., HORN, B., KNIGHT, T., SODINI, C., LEE, H., AND POGGIO, T. Analog vlsi systems for early vision

processing. In *Proceedings of the 1992 IEEE International Symposium on Circuits and Systems* (May 1992), pp. 1644–1647.