Application Driven Embedded System Design: A Face Recognition Case Study*

Karthik Ramani, Al Davis
School of Computing, University of Utah
Salt Lake City, UT 84112, USA
{karthikr | ald}@cs.utah.edu

ABSTRACT

The key to increasing performance without a commensurate increase in power consumption in modern processors lies in increasing both parallelism and core specialization. Core specialization has been employed in the embedded space and is likely to play an important role in future heterogeneous multi-core architectures as well. In this paper, the face recognition application domain is employed as a case study to showcase an architectural design methodology which generates a specialized core with high performance and very low power characteristics. Specifically, we create 'ASIC-like' execution flows to sustain the high memory parallelism generated within the core. The price of this benefit is a significant increase in compilation complexity. The crux of the problem is the need to co-schedule the often conflicting constraints of data access, data movement, and computation. A modular compiler approach that employs integer linear programming (ILP) based 'interconnect-aware' instruction and data scheduling techniques to solve this problem is then described. The resulting core running the compiled code delivers a 1.65x throughput improvement over a high performance processor (Pentium 4) while simultaneously achieving an 80x energy-delay improvement over an energy-efficient processor (XScale) and performs real-time face recognition at embedded power budgets.

Categories and Subject Descriptors

C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems; D.3.4 [**Processors**]: Optimizations

General Terms

Algorithms, Design, Performance

Keywords

embedded systems, domain specific architectures, face recognition, compilers, instruction scheduling, workload characterization

*This work was supported in parts by NSF grant 0541009 and a University of Utah Graduate Research Fellowship.

CASES'07, September 30–October 3, 2007, Salzburg, Austria.

1. INTRODUCTION

Due to the emergence of power consumption as a first order design constraint, embedded and mainstream microprocessors are now faced with the problem of providing a significant increase in performance without a commensurate increase in power or energy consumption. The need for increased performance at reduced power levels has primarily motivated the recent trend towards multi-core processors, e.g. Intel Core Duo, AMD Opteron, TI OMAP. The problem is further exacerbated in the embedded domain where a significant component of the application mix consists of rapidly evolving complex streaming media applications which are inherently real-time programs. Speech recognition, face recognition, and wireless cellular telephony algorithms fall into this application space. Typically, they consist of a sequence of compute-intensive phases (kernels) connected by relatively short *setup* phases.

While general purpose processors (GPPs) are well suited to execute sequential *setup/control* codes, application specific integrated circuits (ASICs) are often used for compute intensive kernels to meet the power and performance constraints of embedded systems. ASICs are highly specialized fixed function devices and therefore complex application suites require multiple ASICs. They are expensive and time-consuming to design, and their inflexible nature implies redesign if the algorithms change. This is problematic given the dynamic nature of embedded system algorithm development. Using digital signal processors (DSPs) or general purpose processors (GPPs) handles the ASIC inflexibility problem but often fails to meet both performance and power constraints. We believe that the keys to solving this dilemma are specialization and parallelism while retaining flexibility through programmability. We call such devices *domain specific architectures* (DSAs).

This paper presents the design of a DSA (ArcFace) specialized for the face recognition domain. Human face recognition is a complex task given the diverse range of facial features and skin color variations. The face recognition domain includes all the processes involved in real time face recognition including flesh toning, segmentation, face detection, and face identification (often referred to as recognition in literature). These processes are generalized object recognition methods and can be adapted to perform other visual recognition tasks. To increase the algorithmic diversity of the domain, we evaluate two fundamentally different techniques for face identification. A detailed characterization (Section 2) of the compute, control, and data access characteristics of all the kernels is performed to create the ArcFace

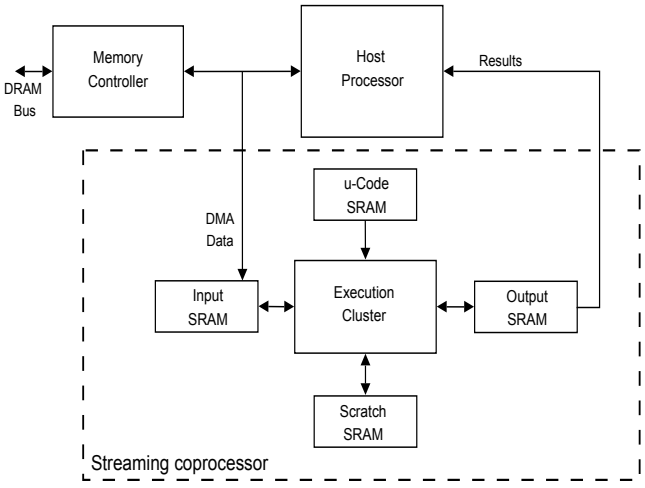

Figure 1: Heterogeneous Multiprocessor Organization

DSA. To our knowledge, this is the first study that compares and contrasts two different face recognition algorithms with respect to their computational complexity and architectural needs.

The memory architecture of the DSA (section 3) is designed to support the data access, communication, execution unit and control characteristics in the face recognition suite. Specifically, our memory system consists of hardware support for multiple loop contexts that are common in the face recognition suite. In addition, the hardware loop unit and address generators provide sophisticated addressing modes which increase IPC since they perform address calculations in parallel with operations performed in the execution units. In combination with multiple dual-buffered SRAM memories, this results in very high memory bandwidth sufficient to feed the multiple execution units.

The architectural model is effectively a long word (VLIW) approach but each bit in our program word directly corresponds to a binary value on a physical control wire. This very fine grained approach was inspired by the RAW project [26]. This allows multiple execution units to be chained together to provide "ASIC-like" computation flows by controlling data movement through the communication fabric between execution units, pipeline registers, and the global interconnect. The result is a programmable DSA (Section 3) whose energy-delay characteristics approach that of an ASIC while retaining most of the flexibility of more traditional programmable processors. Figure 1 illustrates the complete system architecture. The heterogeneous system consists of a GPP that executes the sequential *setup* code while the DSA performs kernel acceleration. Other work [15, 10] has demonstrated the effectiveness of a similar approach for the speech recognition and the wireless telephony domains. However, these studies required that the architectures be manually scheduled at the machine language level. In this work, we address the compilation problem as well.

Program scheduling for this architecture is a complex task for several reasons. The program is effectively horizontal microcode which requires that all of the control points (register output or load enables, execution opcodes, multiplexer select lines, address context updates, etc.) be concurrently scheduled in space and time to create efficient and highly parallel and pipelined flow patterns. To solve this problem, we have created the CoGenE compiler that employs Integer Linear Programming (ILP) based interconnect-aware scheduling techniques to map the kernels to the DSA. In summary, this paper makes the following contributions:

- We perform workload characterization for the face recognition application domain and analyze two different face identification applications: The Elastic Bunch Graph Matching (EBGM) algorithm and the Principle Component Analysis-Linear Discriminant Analysis (PCA/LDA) algorithm, and describe how domain analysis drives the design of the Face Recognition DSA.

- We present the CoGenE compiler framework that employs ILP based interconnect scheduling to map the face recognition applications to the DSA, which solves the programming complexity problem.

The result is an application driven design methodology capable of creating domain specific accelerators which exhibit superior power/performance characteristics.

2. FACE RECOGNITION DESCRIPTION

While the importance of face recognition has motivated numerous algorithms and recognition accuracy evaluation efforts [17], we are particularly interested in face recognition using cheap, low-resolution cameras compatible with low cost embedded systems. Images may be poorly lit, contain occlusions, and may not contain frontal views. Figure 2 shows the major steps involved in our face recognition system. The input for our system is a stream of 320x200 pixel frames arriving at a rate of 5-10 frames per second. The stream is processed one frame at a time and we maintain state to perform motion tracking. The process is a pipeline of kernels, and the goal is to process them in real time.

From a high level perspective, face recognition can be viewed as two sequential phases: face detection and face identification. After initial preprocessing, face detection analyzes video or camera frames to produce a set of normalized skin-tone patches which likely contain a face. Eye location pinpoints the probable eye location candidates and normalizes the patch to meet the Face Recognition Technology (FERET) [17] normalization requirements. It also creates a boundary description for the patch. Face identification then tries to match the probable facial patch to a face in the database. The goal is to minimize the number of false positives and negatives.

The CSU face recognition group has analyzed a variety of face identification algorithms [4]. We choose two algorithms (PCA/LDA and EBGM) from their evaluation suite due to their superior recognition accuracy and relatively high computational parallelism. The PCA+LDA algorithm recognizes faces by performing holistic image matching while the EBGM algorithm compares known features (eyes, nose, etc.) of different faces. Because of the fundamental difference in the two algorithms, the execution, data access and control flow patterns are diverse. We now provide a brief description of the different components in a complete face recognition system followed by a study of the execution characteristics of the system and its memory requirements.

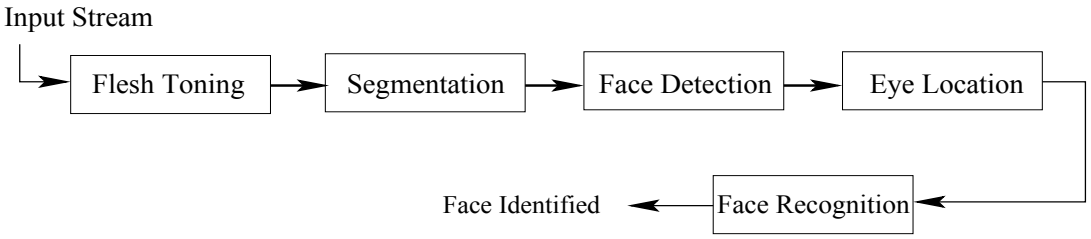

Figure 2: Processing Kernels in a Face Recognition System

Preprocessing: Flesh Toning and Segmentation.

Flesh toning looks for patches of skin toned pixels. Skin colors are more tightly clustered in the HSV color space. Pixels are therefore converted from RGB space to the HSV color space. To improve accuracy, we employ the consensus of two separate flesh toning algorithms based on the Normalized Color Coordinated and the HSV color spaces respectively [14]. The output of this stage is a bit mask of the image marking where the pixel color is a viable flesh tone.

Image Segmentation is the process of clumping together individual pixels into regions where the face might be found. Because our face detection mechanism requires rectangular regions for its operation, we perform two simple mathematical operators: erosion and dilation. An erosion operator examines each pixel and blacks it out unless all its neighbors in a 3x3 pixel map are set [9]. This removes small occlusions and dilation then lights up the pixel if any of its neighbors in a 4x4 window are set.

Viola-Jones Face Detection.

The face detector phase is based on the Viola-Jones approach which is a variant of the AdaBoost algorithm [22, 25]. The AdaBoost strategy is to employ a series of increasingly discriminating filters so that weaker/faster filters need to look at larger amounts of data and the stronger/slower filters examine less data. Viola-Jones takes a similar approach but rather than cascading filters, their approach is to use multiple parallel weak filters to form a strong filter. Viola-Jones achieves a 15x speedup over the Rowley detector [21]. The Viola-Jones code is proprietary but the algorithm was published and a version of this algorithm was developed at the University of British Columbia (UBC). We modified the UBC code to suit our stream based approach. The AdaBoost algorithm also provides statistical bounds on training and generalization errors. Common operations are sum or difference operations between pixels in adjacent rectangular regions. Face detection involves computing the weighted sum of the chosen rectangles and applying a threshold. A 24x24 detector is swept over every pixel in the image and the image is rescaled. A detection will be reported at several nearby pixel locations at one scale and at corresponding locations in nearby scales. We then employ a simple voting mechanism to decide the final detection locations. In our approach, we use a detector which employs 100 different matching criteria.

Holistic Face Recognition: PCA+LDA algorithm.

Our PCA based face recognition algorithm is based on [28]. We chose this algorithm over the Eigenfaces technique [14] due to the increased recognition accuracy in the original FERET study. In the first step, we build a PCA subspace where the face images are projected onto a feature space defined by the eigenvectors of a set of faces. The LDA algorithm is then employed to perform image classification. All the training images from the PCA subspace are grouped according to subject identity and basis vectors are computed for each subject. A test image is then projected onto the PCA+LDA subspace and two distance measures are calculated between each pair of images. The distance measures are then used to label the test image for comparison with known persons in the database.

Topology based Face Recognition: EBGM algorithm.

The EBGM algorithm works on the premise that all human faces have a topological structure and was originally developed by the USC/Bochum group [27]. Faces are represented as graphs, with nodes positioned at facial features such as eyes, nose, etc. and the edges are represented by distance vectors. Distances between the nodes are then used to identify faces. The computational complexity of the algorithm is dependent on the number of feature nodes to be compared. We use a re-implementation of the EBGM algorithm provided by the CSU research group [4]. The EBGM advantage is that it performed well in the original FERET studies on facial images that were not frontal views.

The output of eye location is normalized, smoothed, and rescaled to increase the efficiency of landmark localization in the face recognition step. The normalized image and the landmark locations are used to create face graphs for every image in the database. The final step is to produce a distance matrix for the images. Face identification is based on nearest neighbor classification. In the original CSU implementation, real-time performance was not a goal. Hence, our version employs sufficient code motion and reordering to process the image on a real-time frame-rate basis.

2.1 Workload Characterization

Figure 3 shows the relative execution profiles for the face recognition system with the PCA/LDA and the EBGM algorithms respectively. The native profiling results were obtained using SGI SpeedShop on a 666 MHz R14K processor. The face detection kernel accounts for more than 50% and face identification consumes 25% of the total computation cycles. This implies that detection and identification (PCA/LDA and EBGM) are the most time-intensive kernels and are therefore, the key targets for acceleration.

Memory Characteristics.

Memory and execution characteristics studies are based on the SimpleScalar [2] simulation framework with architec-

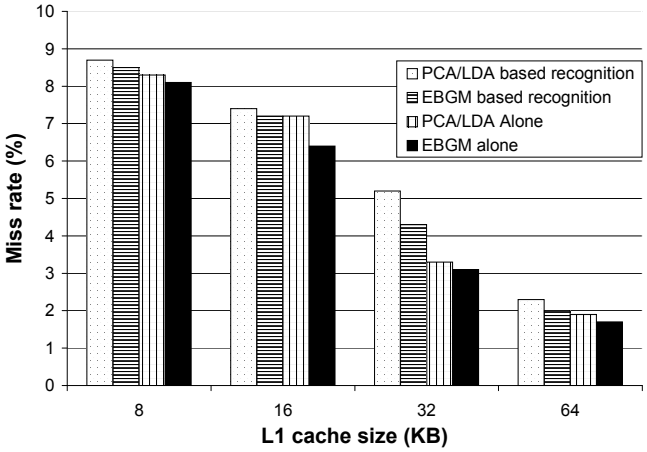

Figure 4: L1 cache miss rates

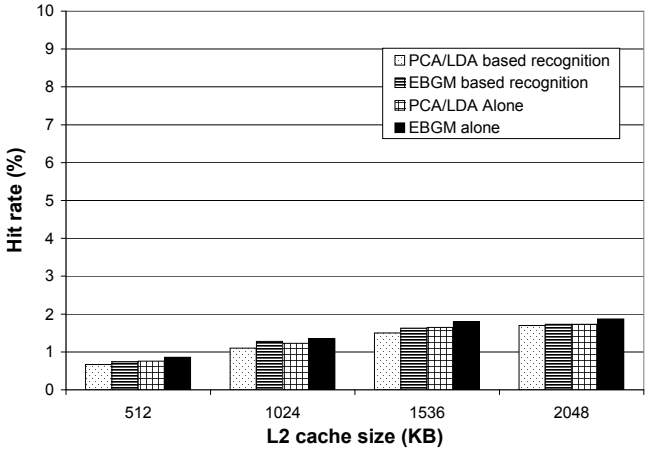

Figure 5: L2 cache hit rates

tural parameters chosen to model an out of order processor (1.7 GHz) similar to a Alpha 21264. We simulate a baseline machine with four integer and floating points units each in order to provide sufficient execution resources, a 2MB L2 cache, and a 600 MHz DRAM interface. In addition, we vary the size of the caches and the number of integer and floating point units for sensitivity analysis.

Figure 4 shows the L1 data cache miss rates for four different configurations: i) complete detection pipeline with PCA/LDA identification, ii) complete detection pipeline with EBGM identification, iii) PCA/LDA face recognition without detection, and iv) EBGM recognition without detection. All the configurations achieve 99.4% hit rates in the ICache. We observe good cache locality for all configurations with a small 8KB data cache which indicates that small self-managed SRAMs are likely to be a good fit for these codes. A 320x200 pixel color image is 188 KB in length while the corresponding gray scale version is about 64 KB. While the image will not directly fit in the L1 cache, the flesh toning kernel requires only one pass over every pixel and hence, data can be accessed in a stream based manner. This provides a 64 KB bitmap image that is processed in at most two passes in the segmentation phase. Good cache locality results because the phase accesses at most two rows at a time. Face detection and recognition kernels process even smaller windows (50x50 pixels or 2.5 KB) on this data multiple times and good cache locality is observed for the whole system. Figure 5 shows the L2 cache (unified) hit rates for the same configurations. We measure the L2 hit rates as the number of hits in the L2 cache divided by the total number of hits for the application. The very low hit percentages suggest that an L2 cache will be prohibitive in terms of energy and area while providing minimal performance improvements.

IPC Saturation.

While the cache behavior of the domain seems to be a good match for embedded processors with limited cache resources, the performance numbers seem to indicate a different view. Table 1 shows the instructions committed per cycle (IPC) for four different configurations as we increase the number of integer and floating point function units. It can be observed that adding more functional units does not provide a commensurate increase in performance. The configuration with 4 integer and floating point units outperforms the one with 2+2 units by a marginal 5%. In addition, we observe a saturation of IPC beyond six units (3+3). Table 2 shows the speedup or slowdown of the four configurations over actual real time corresponding to 5 frames per second. It can be observed that executing a complete face recognition application is at least 2 times slower than real time when we have less than 2+2 functional units. At best, the applications run 1.78 times slower than real time by adding more resources. Executing the identification algorithms alone can achieve real time performance with sufficient resources. The performance improvement comes at the cost of a significant increase in power dissipation. The power dissipated by an out of order core like the Alpha is likely in tens of watts and this exceeds the power budgets available for embedded systems. This motivates the search for a non-GPP approach to provide real-time face recognition at power levels compatible with the embedded space.

There are four reasons for the low performance. First, the face recognition kernels perform a lot of computations of the form $Z[i] = Z[i-1] + \sum_{j=0}^{m} X[j] * Y[W[j]]$ and this introduces loop carried dependencies. Second, the problem is further exacerbated in multi-level loops where such computations entail complex indirect accesses. Third, a large number of loop variable accesses compete with the actual array data accesses, causing port saturation in the data cache. Since the ratio of array variable accesses is high compared to the number of arithmetic operations, contention is a big issue. Finally, the slow real time rate indicates that instruction throughput is low. Even when functional units are available, dependences and memory contention significantly reduce the actual IPC.

2.2 Architectural Implications

Increasing the number of SRAM ports in the system can address the problem of port saturation. However, multiple ports increase the access time, area, and power consumption of the SRAM block. Given that an 8KB cache provides good locality in a conventional cache-based system and the L2 miss rate is high, this motivates a choice to use self managed SRAMs. We chose to use three distributed 8KB SRAMs (input,output, and scratch) for the ArcFace DSA. The input and output SRAMs can be double-buffered to allow simultaneous communication with the host and the execution cluster. The scratch SRAM is used for holding

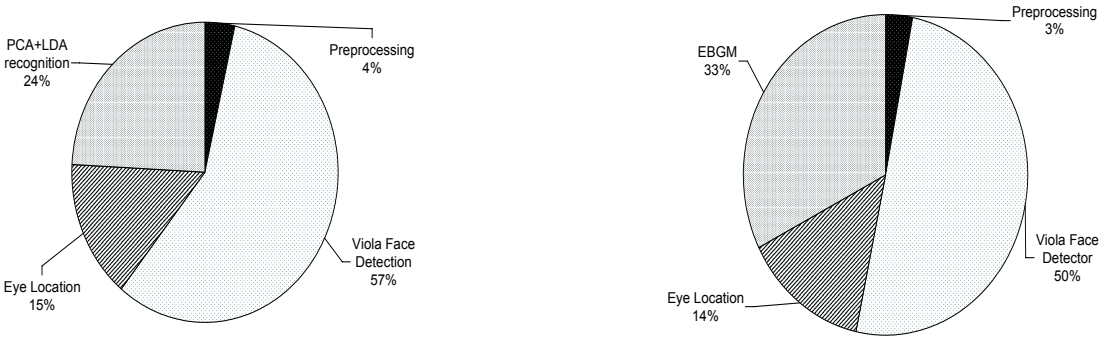

Figure 3: Execution profile for PCA/LDA and EBGM face recognition systems

Number of Execution Units	PCA/LDA complete	EBGM complete	PCA/LDA alone	EBGM alone
1+1	0.651	0.623	0.780	0.757
2+2	0.703	0.683	0.830	0.793
3+3	0.727	0.712	0.897	0.877
4+4	0.729	0.720	0.905	0.890

Table 1: Instructions per Cycle (IPC) for baseline alpha configuration with varying number of execution units

intermediate data. In addition, each SRAM is dual ported to support the needs of the multiple execution units. The system mimics a distributed 24KB cache with 6-ports but does so more efficiently in terms of area, power, and latency.

Multi-level Loop based Addressing.

As with most real-time applications, face recognition loops run for a fixed number of iterations and loop indices are used in data address calculations. The predominant data access pattern consists of 2D array and vector accesses. Extracting parallelism across multi-level nested loops requires complex addressing modes. We employ a loop unit, a programmable hardware structure that provides support for multiple simultaneous loop contexts for efficient data access. The loop unit automatically updates the loop nest indices in the proper order and our implementation is similar to [15]. The Viola/Jones detection kernel requires a maximum of three simultaneous loop contexts. Hence, the ArcFace loop unit supports 3 contexts. Increasing the number of contexts further increases the area, complexity, and power dissipation while providing little performance improvements for the face recognition domain. In addition, the loop unit provides hardware support for modulo scheduling.

Sophisticated Addressing for Memory Parallelism.

The problem of contention between address calculations and actual data computations is only partially solved with distributed memory. The use of programmable Address Generator Units (AGUs) on each SRAM port allows multiple address calculations to be done in parallel with arithmetic operations which improves IPC. Each AGU effectively services the needs for a particular execution unit. The AGUs use the index values provided by the loop unit to facilitate data delivery to the execution units. Hence, the memory system for our DSA consists of a loop unit, three distributed 8KB SRAMs with two ports each, and associated AGUs. This system provides very high memory parallelism by decoupling address and data computations, reducing port saturation, and provides hardware support for the compiler to restructure loops and handle loop carried dependencies.

Execution Back-end: 'ASIC-like' flows.

In a traditional super-scalar processor, instructions are fetched, decoded, issued and retired. Function units receive operands from a register file and return results to the register file. This represents a huge amount of overhead which then gets amortized over a very small piece of function unit work. The challenge is to amortize the overhead over more work in order to increase performance and reduce power consumption. ASICs are complex computational pipelines which transform input data into results with almost no overhead but they lack generality and flexibility. Our execution back-end mimics the ASIC approach while preserving programmability. The use of programmable multiplexers allows function units to be linked into 'ASIC-like' pipelines which persist as long as they are needed. The outputs of each MUX stage and each execution unit is registered which allows value lifetime and value motion to be under program control. This removes the need for a large multi-ported register file which saves significant power with no reduction in performance. Flexibility is preserved by providing the ability to specify interconnect routes via MUX configurations under program control.

The execution resources need to support a large amount of floating point calculations in the face recognition kernels. In addition, integer arithmetic is also required to support address calculations in cases where the AGUs cannot handle these duties autonomously. Our execution units comprise four floating point units and three integer functional units. As will be seen, this provides a good balance between performance and energy consumption.

SIMD vs VLIW trade-offs.

A SIMD approach also delivers high data parallelism and reduces register file complexity by clustering the register file and thereby reducing port complexity. Our VLIW approach provides high instruction level parallelism by performing memory operations and data computations simultaneously, albeit with a larger control overhead due to the width of the instruction word. Our execution back-end is less dependent on a centralized register file. Moreover, the vast difference

Number of Execution Units	PCA/LDA complete	EBGM complete	PCA/LDA alone	EBGM alone
1+1	2.310	2.560	1.530	1.610
2+2	2.050	2.107	1.378	1.383
3+3	1.800	1.870	1.040	1.160
4+4	1.780	1.784	0.978	1.003

Table 2: Speedup/slowdown over real time corresponding to 5 frames per second (real time is scaled to 1

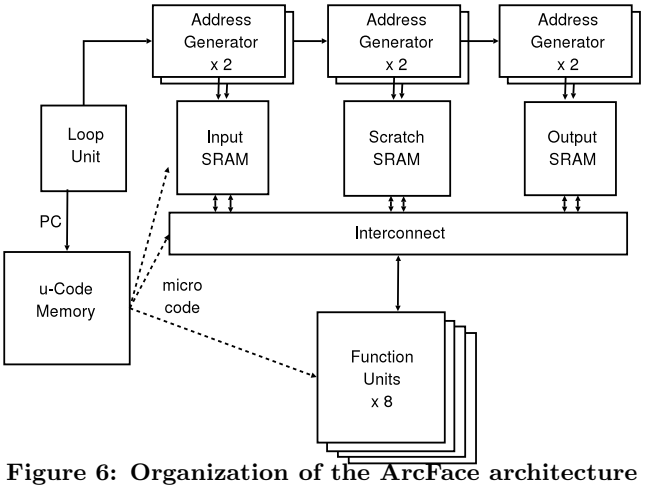

Figure 6: Organization of the ArcFace architecture

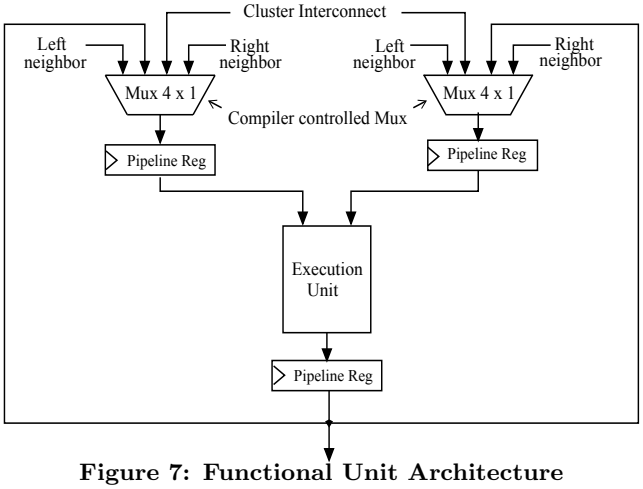

Figure 7: Functional Unit Architecture

in the type of data and address computations performed in a cycle in the face recognition domain makes the SIMD approach less efficient. From performance and energy perspectives, a VLIW approach is more beneficial and is our choice for ArcFace.

3. THE ARCFACE DSA

The face recognition DSA (ArcFace), is shown in figure 6. The memory system includes a loop unit, three 8KB dual-ported and double buffered SRAMs, and six address generator units (AGU). A cluster-wide interconnect is constructed from several layers of multiplexers and connects the memory system and the execution cluster. The execution resources are a cluster consisting of 8 clock-gated function units. These include 4 floating point units, 3 integer units, and a register file. Local bypass paths are provided between neighboring function units. The function unit is illustrated in Figure 7. This shows the inherent pipeline structure where combinational logic is separated by registers. Each execution unit is an arithmetic unit or possibly a register file. Arithmetic units could be internally pipelined.

Address generation, loop control, and multiple execution units all operate concurrently under program control. The compiler generated microcode controls data steering, clock gating (including pipeline registers), and function unit utilization, while permitting single-cycle, program-controlled, reconfiguration of the address generators associated with the SRAM ports. The general result is a cluster that is tailored to the face recognition domain and supports multiple applications, application phases, or interleaved phases of a single pipelined application. Energy efficiency is primarily due to: minimized communication, activity, overhead, ASIC-like pipeline flows, and fine-grained clock gating.

The Potential for Memory Level Parallelism.

The loop unit and the AGUs drive the SRAMs to facilitate efficient communication with the execution cluster. We now evaluate the performance of the memory system. Figure 8 compares the IPC of the baseline alpha machine with different ArcFace configurations: i) ArcFace with perfect back-end implies no stalls due to communication or execution resources, which shows the performance of the memory system, ii) ArcFace with perfect memory system, which indicates the performance of the interconnect and execution cluster back-end, iii) baseline ArcFace configuration with actual memory and back-end, but with seven functional units and the register file, and iv) baseline, but with eight functional units and no register file. It can be observed that the ArcFace configuration with perfect back-end provides as much as a 4.5x IPC improvement for face detection, and around a 10x IPC improvement for face identification (EBGM and PCA/LDA) over the Alpha machine. This shows that the memory system reduces port contention significantly and efficiently supports indirect addressing schemes.

The Potential for 'ASIC-like' flows.

The configuration with perfect memory evaluates the cluster back-end in our system and we observe a 3x improvement for face detection and 6.7x improvement for face identification. The advantage comes from exploiting "ASIC-like" flows where scheduling data for high computation to storage ratio sustains the high memory bandwidth inherent in the system. It also serves to demonstrate the effectiveness of the pipelined registers for storing intermediate values.

Actual Baseline Performance Potential.

The last two configurations in figure 8 show the performance of the baseline ArcFace DSA with the actual memory and actual execution cluster. Here, we also compare the performance of the system with and without a register file in order to evaluate the effectiveness of the register file. In addition, the register file is replaced by an integer functional unit to evaluate performance trade-offs. The baseline ArcFace system provides as much as a 2.7x performance improvement for face detection and a 5.5x-5.8x improvement

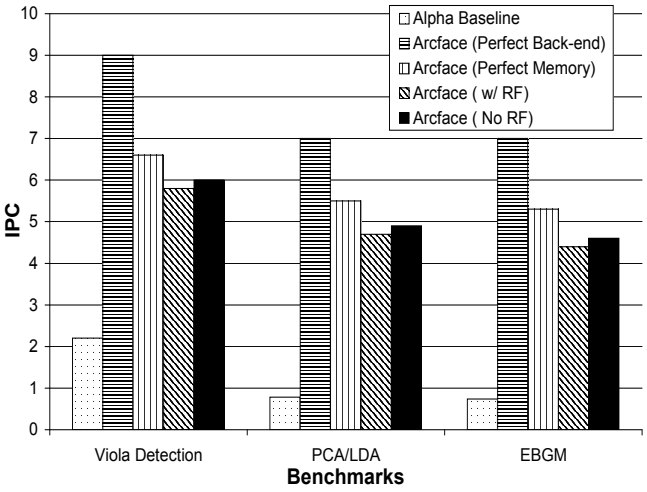

Figure 8: Plots showing the potential for memory parallelism and 'ASIC-like' flows

for the face identification kernels when compared to the Alpha. The execution cluster and memory system are well matched in terms of throughput. The combination of 'high memory parallelism' and 'ASIC-like' flows works well for the face recognition domain. Replacing the register file with an additional integer functional unit provides a marginal 3-4% performance improvement. The register file does ease the difficulty of compiler based scheduling and is a more generally useful structure than another execution unit if the algorithms change in a substantial fashion. Hence, we keep the register file in the subsequent discussion.

Comparing the baseline model to the model with perfect memory shows a performance degradation of about 13-18%. This is explained by the fact that the baseline system employs a cluster-wide interconnect for communication between the memory and the execution units. Due to contention in the global interconnect for data computation and data access, scheduling delays are introduced and we observe a subsequent performance degradation. Employing a hierarchical or separate interconnect will solve the problem, but at increased power costs. Given that we don't need more performance to meet the real time requirements, we make the power conservative choice.

The Compilation Problem.

The fine grained horizontal microcoded nature of ArcFace implies that the compiler is responsible for managing all of the physical resources at an equally fine grained level. Managing different function units, multiple memories and their associated AGUs, and scheduling data flows through the interconnect is a complex task. The inherent programming complexity of the architecture makes hand coding a lengthy and error prone process. Even though the architecture is capable of impressive performance at low power consumption levels, it will be a futile effort unless the scheduling task can be performed automatically by a compiler.

4. THE COGENE COMPILER

4.1 Trimaran to CoGenE

The Trimaran compiler (www.trimaran.org) was the starting point for the CoGenE (Compile Generator Explorer) compiler development. Trimaran was chosen since it allows new back-end extensions, and because its native machine model is VLIW [23]. Significant modifications were needed to transform Trimaran from a traditional cache-and-register architecture to meet the needs of our fine-grained cache-less approach.

The result is a compiler that takes streaming code, written in C, and code generation is parameterized by a machine description file which specifies: the number of clusters, the number and type of functional units in each cluster, the number of levels of inter- and intra-cluster interconnect, and the individual multiplexer configurations. A new back-end code generator that is capable of generating object code for the coprocessor architecture described by the architecture description file was developed. The code generator includes a modified register allocator that performs allocation for multiple distributed register files rather than for a single register file. Since the compiler controls the programming of the multiplexers and the liveness of the pipeline registers, register allocation is inherently tightly coupled with interconnect scheduling. Hence, we introduce a separate interconnect scheduling process after register allocation and our scheduling scheme is based on integer linear programming (ILP) [6] techniques. Before delving into the scheduling details, we provide an overview of ILP based problem solving.

Integer Linear Programming (ILP).

Computing an optimal solution for an ILP program is NP complete [6]. Researchers at Saarland University have contributed to significant advances in improving the efficiency of ILP techniques by reducing the process of enumeration [6]. Integer Linear Programming is the following optimization problem:

$$min\ z_{IP} = c^T x$$

$$x \in P_F \cap Z^n$$

where

$$P_F = \{x | Ax \geq b, x \in IR_+^n\}, c \in IR^n, b \in IR^m, A \in IR^{mxn}$$

The set P_F is called the feasible region and it is integral if it is equal to the convex hull P_I of the integer points ($P_I = conv(\{x | x \in P_F \cap Z^n\})$). In this case, the optimal solution can be calculated in polynomial time, and hence, any formulation of the ILP program should find equality constraints such that P_F is integral.

4.2 CoGenE Compiler Flow

Preliminary Control and Data Analysis.

The overall CoGenE flow is illustrated in Figure 9. The Trimaran loop detection analysis package is used to identify the loops and calculate the start and end conditions. The standard Trimaran data flow packages are used to annotate the dependence graph with variable use and definition locations. Back substitution is then performed to reduce critical path length. After this stage, the number of loops and their characteristics are known.

Modulo Scheduling.

With information from the previous step, we identify the inner most loop and calculate the lowest bound on the initiation interval, similar to the modulo scheduling approach [19].

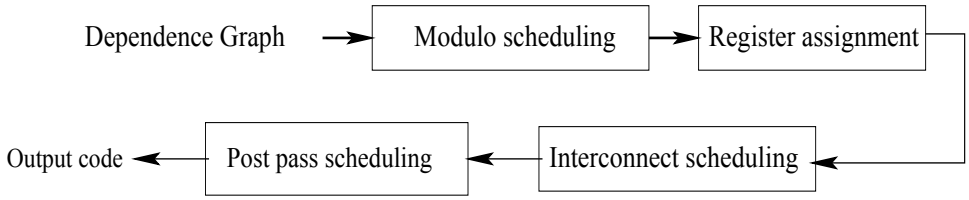

Figure 9: Code Generation

If the bound is high enough to cause degradation, loop unrolling is performed to improve the results of scheduling followed by simple register assignment where the pipeline registers hold the result.

Interconnection Scheduling.

The main decision variables employed are x_{nt}^k where a value of 1 means that instruction n is executed in clock cycle t on execution unit k. The index k of the decision variables is relevant for instructions that can be executed on several different execution units. For all address calculations, the AGUs are paired to a unique execution unit. Let I denote the set of instructions from the input program. Before formulating the integer linear program, we define an interval N(n) which is the earliest control step in which instruction n can be started without violating any data dependencies. The calculation of the interval N(n) is similar to [11].

The scheduling polytope is composed of different types of constraints. The assignment constraint ensures that each instruction is executed exactly once by one execution resource. Let R(n) denote the set of execution unit types that the instruction n can be assigned to:

$$\sum_{k \in R(n)} \sum_{t \in N(n)} x_{nt}^k = 1 \quad \forall n \in I$$

The precedence constraint models the data dependencies within the input program. The dependences can be further classified into two categories: weak- or anti-dependences (Write after Read), and strong dependencies (Read after Write). Write after Write dependencies are not an issue in this architecture since write targets do not conflict. Weak dependencies within a group are allowed. Let w_{mn} represent the minimum number of cycles from start time m to end n during which the dependence is to be respected, then:

$$\sum_k \sum_{t_n \leq t} x_{nt_n}^k + \sum_k \sum_{t_m \geq t - w_{mn} + 1} x_{mt_m}^k \leq 1$$

The precedence constraints exclude any ordering of instructions where data dependences are violated. Until now, the feasibility function is integral, i.e. the solution can be calculated in polynomial time. We now add resource constraints to the system. Resource constrained scheduling is NP complete. Let R_k denote the number of execution units of type k available in the processor. The resource constraint prevents more than R_k instructions being assigned in a cycle. It should be noted that resource constraints also implicitly include the constraints on the multiplexer at the output of the execution units. If U is the pre-calculated upper bound on the number of clock cycles for the input program, then:

$$\sum_{n \in I : k \in R(n)} x_{nt}^k \leq R_k \quad \forall k \wedge 1 \leq t \leq U$$

Now, every integer point saturating the constraints corresponds to a feasible solution of the interconnect scheduling algorithm. The goal is to find a schedule of minimal length L. The value of L is defined by:

$$\sum_k \sum_{t \in N(n)} t x_{nt}^k \leq L \quad \forall n \in I$$

The goal is to minimize the objective function L. So far, our objective function does not take into consideration the instructions that take several clock cycles because of interconnect constraints. This could produce instruction slots with no instructions to be scheduled. The objective function minimizes the execution time as a primary constraint. The ILP model in our infrastructure was solved by the CPLEX solver and the solution was efficiently obtained for most kernels. However, the EBGM face graph recognition took tens of minutes to minimize.

Post Pass Scheduling.

A final pass is done over the code and conflicts in scheduling that can happen due to weak dependencies are distributed to the register file. In addition, those resources that are not used are completely turned off when their instruction slots are empty. For modulo scheduled loops, we check to see if the loop and the address contexts are correctly programmed with the initiation interval.

4.3 Efficiency of 'Interconnect-aware' Scheduling

We now evaluate the efficiency of 'interconnect-aware' scheduling by comparing it against hand-coded schedules. We employ utilization rate, a measure of the total fraction of time for which all the seven functional units in the DSA are employed, as the comparison metric. Table 3 shows that we observe around 62-65% utilization rate for the PCA/LDA and the EBGM face identification kernels. The compiled code achieves an average utilization rate of 60% and achieves 85% of the utilization capability of manual scheduling (utilization rate of 70% for the first four benchmarks). The 15% disparity is because weak dependencies introduce conflicts in scheduling and this causes delays in the compiled code. Further, executing at the 1 GHz target frequency necessitates a longer delay for data transfers across functional units that are farther away from each other. Addressing these issues will improve our scheduling algorithm, however, our technique still delivers a high utilization rate. The high utilization rates also demonstrate the effectiveness of 'interconnect-aware' scheduling for delivering high instruction throughput. We observe tens of seconds of compilation time for all the kernels except for the EBGM kernel in which ILP solving takes a long time to explore a few feasible schedules from a large scheduling space.

Benchmarks	Utilization rate (Compilation)	Utilization rate (Manual)	Compilation time (seconds)
Flesh Tone	0.57	0.74	23
Erode	0.575	0.675	37
Dilate	0.570	0.65	40
Viola	0.69	0.75	60
PCA/LDA	0.62	-	49
EBGM	0.65	-	$\geq$1000

Table 3: Functional unit utilization rate and compilation time for the different face recognition kernels

5. RESULTS

5.1 Experimental Methodology

The Trimaran framework also consists of a cycle accurate simulator which delivers statistics about memory access rates, IPC, throughput in terms of frames processed per second, and the total execution time in cycles. We also include a 32-tap FIR filter to broaden the application domain and to facilitate comparison to an ASIC based design. Our architecture (CoGene compiled code running on the ArcFace DSA) is also compared to three other design options, all of which were normalized to a 0.13μ process :

1. Software running on a 400 MHz Intel XScale processor that represents a highly energy efficient embedded processor. The Xscale does not have floating point instructions, and so, we make our comparisons against an idealized Xscale, where all floating point operations are replaced by integer operations. The code is then run on an actual Xscale processor and performance and power consumption are measured.
2. Software running on a 2.4 GHz Intel Pentium 4 processor that can support the real time requirements of the face recognition kernels.
3. Manually scheduled micro-code implementation running on the simulated cluster architecture representing the best performance point. Energy and performance numbers are calculated using Synopsis Nanosim, a commercially designed spice level simulator, on a fully synthesized and back-annotated Verilog and Module Compiler based implementation. The results are then normalized to a 0.13μ process by employing conservative constant field scaling. The simulated model includes a full clock tree and worst case wire loads based on assigning wire parasitics based on metal 1. Hence, these results are pessimal since in a fabricated design the long wires would be routed on larger metal layers.

To effectively compare the different architectures, we employ throughput measured in terms of the number of input frames processed per second. We employ the energy-delay product as advocated by Horowitz [7] product to compare the efficiency of different processors since both energy and delay for a given unit of work are conflicting constraints for the architect and circuit designer.

5.2 Evaluation

The design goal of the instruction scheduling algorithm is to provide real time performance with minimum energy. In order to evaluate the throughput and energy control capabilities of CoGenE, we compare CoGenE against the performance of hand scheduled code and the Pentium 4. The result is then compared to an XScale based implementation for energy consumption. Finally, we conclude with a comparison of the two face recognition algorithms.

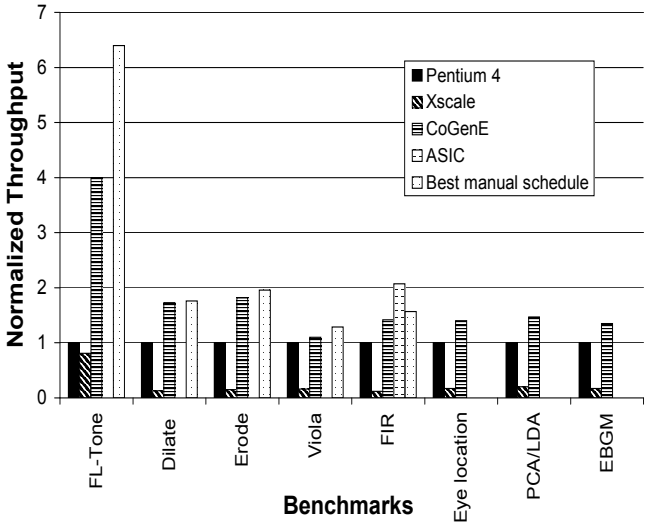

Figure 10: Throughput comparisons for different configurations

Figure 10 compares the throughput (number of input frames processed per second) for the different processors. The hand coded implementation delivers the best throughput. The CoGenE version delivers a throughput that is 1.65 times better than the Pentium 4 processor and 8.64 times better than the XScale processor. This underlines the fact that our CoGenE framework exploits the streaming nature of the face recognition kernels to deliver the throughput necessary to achieve real time constraints. CoGenE is able to achieve 85% of the throughput of manually scheduled code which we feel is satisfactory but also motivates us to further improve our scheduling approach.

Figure 11 and 12 shows the energy consumption per input and the Energy Delay product comparison for the different processors. The CoGenE compiler reduces energy consumption by 9.25x when compared to the low power XScale processor. The energy advantage comes from efficient decoupling between address and data computations provided by the loop unit and AGUs, and by minimizing communication overhead due to the ASIC-like pipeline structures. The result is a DSA that performs face recognition at embedded energy budgets. It is noteworthy the energy-delay product of the Xscale processor is within 35% of the Pentium 4 processor, and that our approach provides 80x improvement over the Xscale. The improvements are consistent across all the applications in the domain.

Kernel Level Analysis.

It is interesting that flesh toning accounts for less than 5% of the total execution time but consumes an incommensurate amount of the total energy. This is because the floating point parallelism in flesh toning exceeds the number of floating point units (four) available in the cluster. This means intermediate results must be saved and retrieved from the register file which is inefficient. The hand scheduled code does a better job of vectorizing the code which indicates that further scheduling improvements are possible. CoGenE does well on the image segmentation phase (erode and dilate kernels) , and the architecture delivers two orders of magnitude better energy-delay product than the XScale.

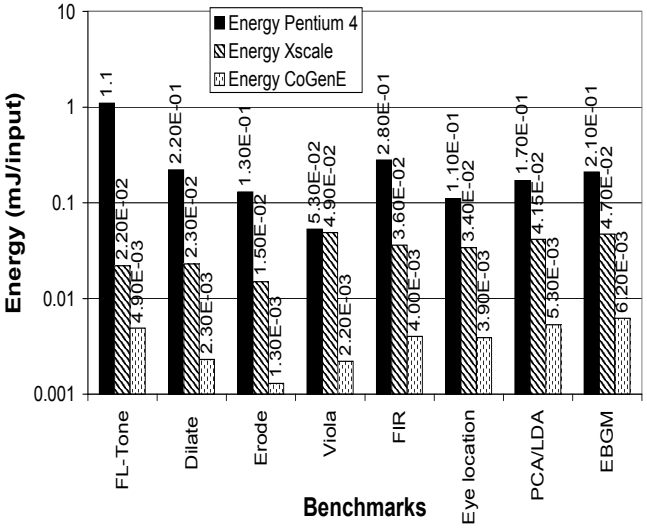

Figure 11: Energy/input packet comparison

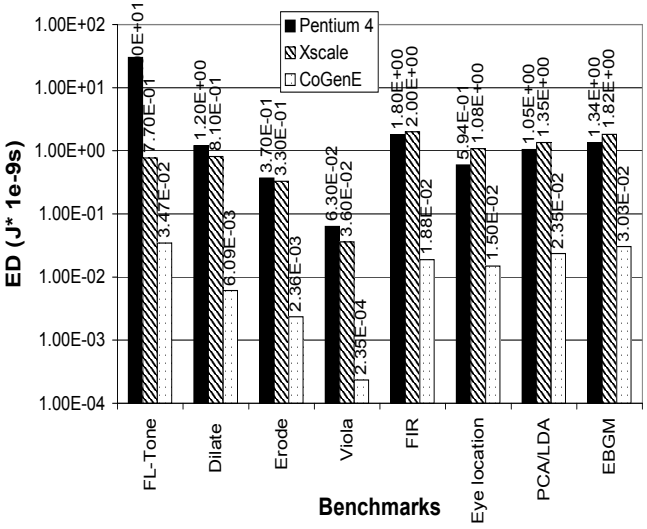

Figure 12: Energy Delay Product Comparison

The Viola/Jones face detection algorithm is characterized by a recurrence that involves two adjacent image rows and an additional row for intermediate for intermediate storage. As the algorithm sweeps over the image, only a 24x24 window needs to be operated on. The algorithm then successively shifts by one pixel position. Pixel value lifetimes are therefore high. The architecture benefits as a result and reduces energy consumption by as much as 22x over the XScale.

The CoGenE FIR version delivers two orders of magnitude energy-delay product improvement over the XScale processor and is only 24x worse than the ASIC implementation. This is partly because the ASIC possesses significantly more functional units than our architecture.

PCA/LDA vs EBGM.

One of the goals of this study is to compare two fundamentally different face recognition algorithms and to identify the algorithm that is better suited for hardware implementation. The PCA/LDA algorithm is a holistic image comparison algorithm as opposed to the EBGM algorithm. The EBGM algorithm requires an additional normalization step after face detection to increase the accuracy of the algorithm. This adds computational complexity in the algorithm and contributes to the 9% performance advantage of PCA/LDA algorithm. The PCA-LDA algorithm also has a 17% advantage in energy and a 30% advantage in energy-delay product. We then reduced the number of facial feature nodes in the EBGM algorithm in order to reduce complexity but found that accuracy immediately fell to unacceptable levels. We conclude that the PCA-LDA algorithm is superior for our architecture and compilation approach.

6. RELATED WORK

Mathew *et al.* [14] perform a detailed characterization of a feature recognition system based on the Eigenfaces algorithm. In contrast, to our knowledge, this is the first study that compares and contrasts the hardware needs of different recognition algorithms. Improving performance or power via VLIW techniques is a common theme in modern embedded systems [1] including mapping and instruction scheduling techniques [13, 24]. However, these efforts do not address low-level communication issues.

High-performance compilation techniques have also been investigated: RAW [12], CGRAs [16], Imagine [20], and Merrimac [5]. The RAW machine has demonstrated the advantages of low-level scheduling of data movement and processing in function units spread over a two dimensional space and this work motivates our fine-grained resource control approach. The main difference is that our approach also tries to minimize energy consumption as a first order design constraint. Mahlke's group has also developed automated techniques for identifying candidate code blocks for coprocessor acceleration and for generating customized instruction set extensions to control those processors [3]. A similar approach by Pozzi also provides graph-based optimizations for micro-architectural constraints such as limited register ports [18]. The main differences between these efforts and our work is that our coprocessor model is more autonomous and attempts to co-optimize performance and energy consumption rather than just performance.

Tensilica's Xtensa system [8], ARM's OptimoDE processor, and IBM's Cell processor are all current commercial approaches in the high performance, energy-efficient embedded systems domain. The main difference is that the user designs a custom VLIW machine by specifying a customized instruction set. Our approach is driven by an application suite and our architecture provides a richer set of options than a traditional more coarse grained VLIW approach.

7. CONCLUSION AND FUTURE WORK

We have presented the workload characterization of a complete face recognition system employing two fundamentally different face identification algorithms. This characterization was then used to create a face recognition DSA using a novel architectural approach, albeit with increased programming complexity. The solution to this dilemma is the CoGenE compiler which employs ILP-based 'interconnect-aware' scheduling schemes to solve the constraints imposed by the highly concurrent scheduling of functional, communication, and storage resources in the DSA. The result is a face recognition system which is capable of delivering more throughput than a Pentium 4 while simultaneously consuming less energy than an XScale. In conclusion, we leverage

architectural design and compilation efficiency to perform real time face recognition at embedded power budgets.

During the course of this effort, we have uncovered numerous opportunities for future research. Register and interconnect scheduling can be done in either order and the second process is limited by decisions made in the first. We intend to investigate an integrated approach. While the CoGenE compiler does a good job of automating our previous biggest problem, e.g. manual instruction scheduling, the end goal of this research is to automate as much of the DSA design process as possible. This implies automating the two remaining manual phases: architecture description creation and splitting the application suite into host and streaming components amenable to DSA acceleration. We believe that the CoGenE compiler can automatically create the architecture description file and subsequently modify it during design space exploration via the simulation infrastructure. Automatic splitting of the original application codes will be a harder task and an interactive tool that significantly aids the process is more likely to succeed.

8. REFERENCES

[1] C. Akturan and M. F. Jacome. Caliber: A software pipelining algorithm for clustered embedded VLIW processors. In *Proceedings of the International Conference on Computer Aided Design (ICCAD)*, pages 112–118, 2001.

[2] D. Burger and T. Austin. The Simplescalar Toolset, Version 2.0. Technical Report TR-97-1342, University of Wisconsin-Madison, June 1997.

[3] N. Clark, H. Zhong, and S. Mahlke. Processor acceleration through automated instruction set customisation. In *Proceedings of MICRO*, 2003.

[4] Colorado State University. Evaluation of face recognition algorithms. http://www.cs.colostate.edu/evalfacerec/, 2003.

[5] W. Dally and P. Hanrahan. Merrimac: Supercomputing with Streams. In *Supercomputing*, 2003.

[6] F. EisenBrand. *Gomory-Chvatal Cutting Planes and the Elementary Closure of Polyhedra*. PhD thesis, Saarland University, 2000.

[7] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. *IEEE Journal of Solid-State Circuits*, 31(9):1277–1284, Sept. 1996.

[8] R. E. Gonzalez. Xtensa — A configurable and extensible processor. *IEEE Micro*, 20(2):60–70, /2000.

[9] R. Haralick and L. Shapiro. *Computer and Robot Vision*, volume 1, chapter 5. Addison-Wesley Publishing Company, 1992.

[10] A. Ibrahim. *ACT: Adaptive Cellular Telephony Co-Processor*. PhD thesis, University of Utah, December 2005.

[11] D. Kastner and S. Winkel. Ilp based instruction scheduling for ia-64. In *ACM SIGPLAN LCTES workshop*, pages 145–154, 2001.

[12] W. Lee, R. Barua, M. Frank, D. Srikrishna, J. Babb, V. Sarkar, and S. P. Amarasinghe. Space-time scheduling of instruction-level parallelism on a raw machine. In *Architectural Support for Programming Languages and Operating Systems*, pages 46–57, 1998.

[13] R. Leupers. Instruction scheduling for clustered VLIW DSPs. In *IEEE PACT*, pages 291–300, 2000.

[14] B. Mathew, A. Davis, and R. Evans. A characterization of visual feature recognition. In *Proceedings of the IEEE 6th Annual Workshop on Workload Characterization (WWC-6)*, pages 3–11, October 2003.

[15] B. K. Mathew. *The Perception Processor*. PhD thesis, University of Utah, August 2004.

[16] H. Park, K. Fan, M. Kudlur, and S. Mahlke. Modulo Graph Embedding: Mapping Applications onto Coarse-Grained Reconfigurable Architectures. In *CASES*, October 2006.

[17] P. Phillips, H. Moon, S. Rizvi, and P. Rauss. The feret evaluation methodology for face-recognition algorithms. In *T-PAMI*, volume 22, pages 1090–1104, October 2000.

[18] L. Pozzi and P. Ienne. Exploiting Pipeline to Relax Register-File Port Constraints of Instruction Set Extensions. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 2–10, September 2005.

[19] B. R. Rau. Iterative modulo scheduling: an algorithm for software pipelining loops. In *Proceedings of the 27th Annual International Symposium on Microarchitecture*, pages 63–74. ACM Press, 1994.

[20] S. Rixner, W. J. Dally, U. J. Kapasi, B. Khailany, A. Lopez-Lagunas, P. R. Mattson, and J. D. Owens. A bandwidth-efficient architecture for media processing. In *International Symposium on Microarchitecture*, pages 3–13, 1998.

[21] H. A. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–38, 1998.

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

[23] M. S. Schlansker and B. R. R. Cover. EPIC: Explicitly parallel instruction computing. *Computer*, 33(2):37–45, 2000.

[24] M. D. Smith, M. Lam, and M. A. Horowitz. Boosting beyond static scheduling in a superscalar processor. In *Proceedings of the 17th Annual Symposium on Computer Architecture*, pages 344–354, 1990.

[25] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Dec. 2001.

[26] E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe, and A. Agarwal. Baring it all to software: Raw machines. *IEEE Computer*, 30(9):86–93, 1997.

[27] L. Wiskott, J. Fellous, N. Kruger, and C. Malsburg. Face Recognition by Elastic Bunch Graph Matching. Technical Report 96-08, Ruhr-Universitat Bochum, April 1996.

[28] W.Zhao, R. Chellappa, and A. Krishnaswamy. Discriminant analysis of principal components for face recognition. In *International Conference on Face and Gesture Recognition*, April 1998.