COGENE: AN AUTOMATED DESIGN FRAMEWORK FOR DOMAIN SPECIFIC ARCHITECTURES

by

Karthik Ramani

A dissertation submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

School of Computing
The University of Utah
SUPERVISORY COMMITTEE APPROVAL

of a dissertation submitted by

Karthik Ramani

This dissertation has been read by each member of the following supervisory committee and by majority vote has been found to be satisfactory.

__________________________
Chair: Al Davis

__________________________
Rajeev Balasubramonian

__________________________
John Regehr

__________________________
Ganesh Gopalakrishnan

__________________________
Paolo Faraboschi
To the Graduate Council of the University of Utah:

I have read the dissertation of ________ Karthik Ramani ________ in its final form and have found that (1) its format, citations, and bibliographic style are consistent and acceptable; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the Supervisory Committee and is ready for submission to The Graduate School.

Date

Al Davis
Chair, Supervisory Committee

Approved for the Major Department

Martin Berzins
Chair/Dean

Approved for the Graduate Council

David S. Chapman
Dean of The Graduate School
ABSTRACT

The embedded system space is characterized by a rapid evolution in the complexity and functionality of applications. In addition, the short time-to-market nature of the business motivates the use of programmable devices capable of meeting the conflicting constraints of low-energy, high-performance, and short design times. The keys to achieving these conflicting constraints are specialization and maximally extracting available application parallelism. General purpose processors are flexible but are either too power hungry or lack the necessary performance. Application specific integrated circuits (ASICS) efficiently meet the performance and power needs but are inflexible. Programmable domain specific architectures (DSAs) are an attractive middle ground, but their design requires significant time, resources, and expertise in a variety of specialties, which range from application algorithms to architecture and ultimately circuit design. This dissertation presents CoGenE, a design framework that automates the design of energy-performance optimal DSAs for embedded systems. For a given application domain and a user chosen initial architectural specification, CoGenE consists of a Compiler to generate execution binary, a simulator Generator to collect performance/energy statistics, and an Explorer that modifies the current architecture to improve energy-performance-area characteristics. The above process repeats automatically until the user specified constraints are achieved. This removes or alleviates the time needed to understand the application, manually design the DSA, and generate object code for the DSA. Thus, CoGenE is a new design methodology that represents a significant improvement in performance, energy dissipation, design time and resources.

This dissertation employs the face recognition domain to showcase a flexible architectural design methodology that creates “ASIC-like” DSAs. The DSAs are ISA-independent and achieve good energy-performance characteristics by co-
scheduling the often conflicting constraints of data access, data movement, and computation through a flexible interconnect. This represents a significant increase in programming complexity and code generation time. To address this problem, the CoGenE compiler employs integer linear programming (ILP) based 'interconnect-aware' scheduling techniques for automatic code generation. The CoGenE explorer employs an iterative technique to search the complete design space and select a set of energy-performance optimal candidates. When compared to manual designs, results demonstrate that CoGenE produces superior designs for three application domains: face recognition, speech recognition and wireless telephony.

While CoGenE is well suited to applications that exhibit a streaming behavior, multi-threaded applications like ray tracing present a different but important challenge. To demonstrate its generality, CoGenE is evaluated in designing a novel multicore $N$-wide SIMD architecture, known as StreamRay, for the ray tracing domain. CoGenE is used to synthesize the SIMD execution cores, the compiler that generates the application binary, and the interconnection subsystem. Further, separating address and data computations in space reduces data movement and contention for resources, thereby significantly improving performance compared to existing ray tracing approaches.
CONTENTS

ABSTRACT ........................................................................... 4
LIST OF FIGURES ................................................................. 8
LIST OF TABLES ................................................................. 9

CHAPTERS

1. INTRODUCTION ................................................................. 1
   1.1 Traditional Approaches and Drawbacks .......................... 3
   1.2 CoGenE: The Grand Goal ........................................... 5
      1.2.1 Brief Overview of Framework ......................... 7
      1.2.2 Evaluation ...................................................... 8
   1.3 Dissertation Statement ........................................... 9
   1.4 Road-map ................................................................. 10

2.RELATED WORK .............................................................. 12
   2.1 Applications ............................................................ 12
      2.1.1 Face Recognition ........................................... 12
      2.1.2 Ray Tracing .................................................... 12
   2.2 Compilers and Scheduling ........................................ 13
   2.3 Embedded Architectures .......................................... 14
      2.3.1 Architectural Support for Ray Tracing ............ 16
      2.3.2 Design Space Exploration .............................. 16

3. FROM APPLICATIONS TO ARCHITECTURE .................... 18
   3.1 Speech Recognition Overview ................................... 18
   3.2 Wireless Telephony Overview ................................... 19
   3.3 Face Recognition System Overview ........................... 20
      3.3.1 Preprocessing: Flesh Toning and Segmentation .... 21
      3.3.2 Viola-Jones Face Detection .............................. 21
      3.3.3 Holistic Face Recognition: PCA+LDA algorithm ... 22
      3.3.4 Topology based Face Recognition: EBGM algorithm ... 22
   3.4 Workload Characterization ....................................... 23
      3.4.1 Memory Characteristics ................................... 23
      3.4.2 IPC Saturation ............................................... 24
   3.5 Architectural Implications ......................................... 26
      3.5.1 DSA memory Architecture ............................... 28
      3.5.2 Execution Back-end: “ASIC-like” flows ............. 29
4. DSA SYSTEM ARCHITECTURE ........................................... 31
   4.1 DSA Evaluation for Face Recognition .......................... 33
REFERENCES ........................................................................ 37
LIST OF FIGURES

1.1 Grand Goal: Automation from apps to chips .......................... 6
3.1 Processing Kernels in a Face Recognition System ....................... 25
3.2 Execution profile for PCA/LDA and EBGM face recognition systems . 25
3.3 L1 cache miss rates .................................................. 25
3.4 L2 cache hit rates .................................................... 27
4.1 Heterogeneous Multiprocessor Organization ............................. 32
4.2 Organization of the recognition DSA .................................. 32
4.3 Functional Unit Architecture ......................................... 35
4.4 Plots showing the potential for memory parallelism and 'ASIC-like'
    flows ................................................................. 35
LIST OF TABLES

3.1 Instructions per Cycle (IPC) for baseline alpha configuration with varying number of execution units (XUs) ................................................................. 27
3.2 Speedup/slowdown over real time corresponding to 5 frames per second (real time is scaled to 1) ................................................................. 27
CHAPTER 1

INTRODUCTION

Embedded systems have revolutionized the way we interact, perceive, and communicate information. The diverse characteristics of such devices have facilitated their deployment in many areas: inexpensive cellular phones for communication and mobile access to information, reliable pacemakers in the field of medicine, security cameras for surveillance purposes, etc. Recent advances have created a strong market desire for information fusion [60], a broad term that refers to a phenomena in which many different technologies are combined to provide the user with a plethora of usage scenarios. For example, a phone may be used to seamlessly switch between different networks without affecting call continuity and clarity. In the future, a single device will be expected to support many different technologies. While designing such systems presents many new problems to system designers, the following challenges create significant roadblocks.

Applications. User demand for complex applications and easy-to-use interfaces drives the embedded application space. Providing natural human interfaces requires support for applications like face and speech recognition [42, 20, 36], real-time graphics [13], etc. For communication, a device needs to support a wide variety of cellular standards [32]. The algorithmic complexity of these applications is growing faster than Moore’s law [60] but, current embedded designs [59] are not flexible enough to adapt to these changes.

Functional fusion, in which one device supports a diverse set of applications, is now a dominant market desire. The iPhone [58] is one example of such a device that provides a few applications including the touch interface, audio playback, etc. In this case, the functionality for each application is provided by employing
a dedicated application specific integrated circuit (ASIC). In the future, devices have to be powerful enough to support an almost ubiquitous set of applications like video gaming, gesture interfaces, and traditional desktop applications. Fusing tens of ASICs in a single device will be practically infeasible. Simultaneously, a user may not need all the applications at the same time. This provides an opportunity to design a few high performance processors that can replace tens of ASICs.

The energy-performance conundrum. The arrival of heterogeneous computing systems like the IBM Cell [28], Intel Larrabee [73] and AMD Fusion [4] has blurred the design requirements in the embedded and desktop computing landscapes. Every new generation of devices is expected to provide an improved level of performance when compared to its predecessor. Power dissipation has also emerged as a first order design constraint. For mobile devices in particular, energy dissipation should be contained with in strict usage requirements [59]. Unfortunately, battery capacities in mobile devices have been projected to improve at a meager 3-7% [60] every year. Given the exponentially increasing algorithmic complexity, this exacerbates the problem of delivering high performance, low energy, and increased flexibility.

Nature of Business. Every new fabrication process requires high initial costs [59, 34] for manufacturing a single chip. Further, millions of chips have to be sold for amortizing the huge capital investments and for obtaining sustainable profits. The market need to support new applications every year mandates very short design times and requires large design teams with a variety of expertise ranging from applications to architecture and circuit design. Thus, the business of embedded devices is governed by extremely short design cycles and economies of large scale [59, 34]. These two constraints are in direct conflict with the amount of time and resources involved in designing and verifying processors in current process technologies. This calls for design methodologies that are scalable and flexible enough to adapt to the volatile markets.

Environment. Depending upon the surroundings in which the system is deployed, various constraints have to be met. Desktop computing allows for sophis-
icated cooling techniques and hence, performance rather than power dissipation is a critical requirement. In contrast, small size is important for mobile devices. Easy to use interfaces and ergonomic style are necessary for cellular phones. The deployment environment thus introduces constraints that further complicates the design of computing systems.

1.1 Traditional Approaches and Drawbacks

Over the last two decades, application specific integrated circuits (ASICs) were predominantly deployed for embedded computing systems due to their fantastic energy-performance characteristics. This worked well for fixed function devices as ASICs provide a high level of functional specialization while being optimized for area, performance, and power dissipation. Supporting a plethora of complex applications in the future will require lengthy design cycles for each ASIC [34]. In addition, changes in the application will incur expensive re-design costs. Digital signal processors (DSPs) and general purpose processors trade-off energy efficiency to provide flexibility in supporting many applications. They employ a general instruction set (ISA) to support any sequence of operations in a program. The side-effect is that they incur a high control and data access overhead to perform the actual computations. The cost of generality is that they cannot meet the performance and energy requirements for certain applications like recognition, cellular standards, and real-time graphics. These applications are characterized by inter-twined sequential and parallel code kernels phases. While GPPs can deliver good performance for control-intensive sequential code, they are either resource limited to deliver high performance or incur too much power for compute-intensive kernels [61]. A good solution is to employ a heterogeneous multi-processor in which the GPP executes the sequential code and the accelerator executes the various kernels. In devices like the iPhone [58], tens of ASICs perform the various kernel processing activities, although not all of the applications run at the same time. Hence, the goal of this study is to employ programmable accelerators to replace tens of ASICs.
This dissertation argues that the keys to solving the often conflicting goals of energy-performance efficiency and application scalability are specialization and parallelism while retaining flexibility through programmability. Such architectures are referred to as domain specific architectures (DSAs). The DSA [47, 32, 61] is specialized to extract the parallelism within the various kernels of an application domain. For example, the face recognition domain includes all the processes involved in real time face recognition including flesh toning, segmentation, face detection, and face identification. A detailed characterization of this domain involves resources and time and is performed in chapter 3. The compute, control, and data access characteristics of all the kernels are analyzed to create a recognition DSA.

The memory system of the DSA consists of hardware support for multiple loop contexts that are common in embedded applications. In addition, the hardware loop unit (HLU) 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 SRAM memories, this results in very high memory bandwidth sufficient to feed the execution units. The program is horizontally microcoded and each bit in the program word directly corresponds to a binary value on a physical control wire. This very fine grained VLIW approach was inspired by the RAW project [79]. The side-effect of this micro-code approach is ISA-independence and provides the flexibility to mimic the data flow and operations within the program closely while incurring minimal overhead. Multiple execution units can be chained together to provide "ASIC-like" computation flows due to program controlled data movement through the DSA’s resources rather than the usual fetch, decode, and execute micro-architecture. The result is a programmable "ASIC-like" DSA whose energy-delay characteristics approach that of an application specific integrated circuit (ASIC), while retaining most of the flexibility of more traditional programmable processors.

The cost of this micro-code approach is increased compiler complexity due to the need to schedule data movement, memory access, register allocation, and execution unit utilization on a cycle by cycle basis. Compile time is also problematic.
[49, 61], although the compile-rarely nature of these systems mitigates this issue. Another drawback is that it incurs significant time and resources to understand the application domain and design the DSA. For example, the face recognition approach [61] involved man-months of characterization, manual code generation, and architecture design time. Finally, the design of programmable DSAs requires expertise in a variety of specialties, which range from application algorithms to architecture and ultimately circuit design. To solve these problems, this dissertation presents and explores CoGenE, a single unified framework that automates the design of DSAs for various application domains. The goal is to reduce capital costs, time, and resources significantly while meeting the often conflicting system design goals.

1.2 CoGenE: The Grand Goal

CoGenE, which stands for Compiler-simulator Generator-design Explorer, is a toolkit intended for use by application experts. The complete automation flow is shown in figure 1.1. The expert factors the application into sequential code that runs on the general purpose host and kernel code that runs on the DSA. In adherence to the stream model employed by the framework, the kernel code is modified manually to process data on a per-frame basis and represents streaming code in C. This code is fed as input to CoGenE. The framework takes the suite that defines the domain, an initial architecture specification based on our DSA approach, and generates a simulator, and an executable binary for that architecture. The architecture is then simulated and both energy and performance statistics are cataloged. Simulation and compiler data affecting area, energy and performance are then combined. The architecture description is then modified to better satisfy user specified constraints for any combination of area, power/energy, and performance at which point the process repeats. Finally, the user is given a set of feasible design points that satisfy his/her requirements. Results ?? demonstrate that automation works independent of the choice of the initial starting point and hence, requires little or no architecture expertise from the application expert. In cases where the
Figure 1.1. Grand Goal: Automation from apps to chips
compiler is not able to generate a feasible code schedule, a choice is given to the application expert to either revisit the code-factoring process or to employ another architectural choice for exploration. In general, CoGenE removes or alleviates the need for compilation, circuit, and architecture expertise, and the error prone process of designing a specialized accelerator for a given application suite. It also evaluates many more design options than would be possible without similar automation. The result is improved design quality and a significant reduction in design time.

1.2.1 Brief Overview of Framework

CoGenE integrates three distinct activities that all contribute to the design process: the compiler that generates execution binary given an input architecture template, the simulator generator that creates a cycle-accurate simulator for this template and collects statistics, and the design explorer that explores the architectural design space to arrive at energy-performance optimal designs.

The DSA approach employed in CoGenE chains multiple execution units to mimic the data computations within the application at very low overhead. While this delivers high performance, this requires simultaneous scheduling for data motion, function units, and memory accesses in both space and time. To solve this problem, the CoGenE compiler employs integer linear programming (ILP) based interconnect-aware scheduling techniques to map the kernels to the DSA. The code optimization tactics are based on [61, 62, 26] which have shown that interconnect-aware scheduling improves performance and energy dissipation. After preliminary control and data flow analysis is performed, the inner most loop is identified and memory addressing is set up. After register assignment, ILP based interconnection scheduling is done followed by post pass scheduling to resolve conflicts.

The resulting object code from the compiler is executed on the simulator. Performance statistics are collected from the resulting compilation and simulation schedules. Energy dissipation is estimated using high level parameterizable power models. Power models employ analytical models from Wattch [10] for all predictable structures and empirical models similar to [65, 66] for complex structures like
the ALU and interconnects. Area estimates are obtained from Synopsys MCL and design compiler. CoGenE’s exploration phase then analyzes these statistics to identify potential architectural options for performance or energy improvements.

The simple iterative design space exploration algorithm (chapter ??) is based on analyzing the source of performance problems that appear during compilation and simulation. Stall causes such as insufficient parallelism, routability problems, etc., all boil down to usage conflicts for various physical resources in the architecture. Adding the appropriate resources (a process called dilation) will improve performance but will also increase area and energy consumption. Appropriate removal of lightly or unused resources (thinning) may reduce performance but will also reduce energy and area. Improper dilation will increase energy with no performance benefit and improper thinning will significantly reduce performance with little energy benefit. During diagnosis, several options are investigated to remove the bottleneck (this term is used in the context of area and energy as well as the more common performance usage), and each option is assigned a cost. In addition to maximizing performance [23], the notion of cost attempts to optimize energy dissipation and compilation complexity. The least cost alternative is tried first. The process iterates and results in near-optimal designs for user specified energy-performance constraints.

1.2.2 Evaluation

Effectiveness of CoGenE is evaluated as a case study for three important application domains: face recognition, speech recognition, and wireless telephony. These domains are fundamentally different in their access, control, and computational characteristics [61, 47, 32] and present a diverse embedded workload [42, 20]. The results demonstrate that CoGenE arrives at designs that are competitive with or better than previous best-effort manual designs and significantly better than what can be obtained on more conventional programmable platforms such as the Xscale. The CoGenE compiler generates efficient schedules for a variety of architectures within the DSA framework and performance approaches that of the best manual
schedules. The side-effect is that automatic compilation removes the need to invest man-hours into manual code generation. The exploration process is independent of the choice of the initial architectural template and results show that CoGenE always arrives at optimal energy-performance candidates in a very short time. Overall, this design tool can be employed by application experts to design optimal energy-performance DSAs with little or no expertise in the area of embedded system design.

DSAs designed for embedded applications demonstrate the robust nature of CoGenE for stream-oriented workloads. Workloads that are multi-threaded by nature represent a different test to the framework. To evaluate the generality of CoGenE, this dissertation analyzes its capability on ray tracing, a multi-threaded graphics application. Ray tracing was chosen due to it many applications in entertainment, science, and industry. In addition, designing an architecture for ray tracing has implications beyond embedded computing.

To fit the CoGenE streaming model, stream filtering is employed. This approach [26] that recasts the basic ray tracing algorithm as a series of filter operations that partition an arbitrarily sized group of rays into active and inactive subsets in order to exploit coherence and achieve speedups via SIMD processing. CoGenE is employed to design various constituent parts of StreamRay [64, 63], a novel multi-core architecture that efficiently supports ray tracing. The architecture consists of two major subsystems: the ray engine, which performs address computations to form large data streams for SIMD processing, and the wide-SIMD filter engine, which performs the data and filter computations. CoGenE is employed to synthesize the filter engine and the interconnect subsystem. The compiler also generates code for the filter engine. Results demonstrate that StreamRay improves performance significantly and delivers interactive frame rates of 15-32 frames/second (fps) for scenes of high geometric complexity.

1.3 Dissertation Statement

Given the rapidly evolving application space, the keys to meeting the conflicting
constraints of high performance, low energy dissipation, flexibility, and extremely short design time are specialization and maximally exploiting available application parallelism. This dissertation provides the following contributions in achieving these goals:

- **CoGenE.** A unified design framework that analyzes an application domain and presents a set of energy-performance optimal DSAs automatically to the application expert who has little knowledge of architecture and circuit design. CoGenE also provides an optimizing compiler that automates code generation. By automating the process of workload characterization, compilation, and architectural design, CoGenE represents a new design methodology that delivers a significant improvement in system performance, power dissipation, resources, and design time.

- **Workload Studies.** During the initial stage of CoGenE development, the face recognition domain was completely characterized to design a recognition DSA. This is the first study that analyzes the computational requirements of many different face recognition algorithms.

- **CoGenE for ray tracing.** This dissertation presents StreamRay, a novel multicore architecture that efficiently supports ray tracing. The CoGenE compiler generates object code for the various ray tracing kernels. The CoGenE explorer was also employed to automatically synthesize the SIMD execution cores and the interconnection subsystem. Given the importance of this emerging application and the new challenges this domain presents to CoGenE, our results demonstrate the robustness of CoGenE in designing DSAs for a variety of compute intensive applications. It also opens a novel area for future work.

1.4 Road-map

A survey of the background work and its limitations is performed in chapter 2. Chapter 3 showcases our DSA design approach by systematically characterizing and
analyzing the face recognition domain. Chapter 4 discusses the various features of our "ASIC-like" DSA methodology followed by the compilation methodology in chapter ???. Design space exploration is explained in chapter ???. The evaluation infrastructure and results are discussed in chapters ?? and ?? respectively. Ray tracing and DSA design is presented in chapter ???. Conclusions and future work are summarized in chapter ??.
CHAPTER 2

RELATED WORK

Embedded designs have to achieve the often conflicting goals of high performance, low power, flexibility, and short design time. In recent years, contributions have been made to meet some or all of these goals. The CoGenE design methodology is compared and contrasted against various approaches in the literature to showcase the major differences. It also helps to highlight the novel capabilities provided by CoGenE.

2.1 Applications

2.1.1 Face Recognition

Gottumukkal [25] designed a FPGA based face recognition (identification) architecture that identifies a person from a database. The difference is that this dissertation performs the study of a complete face recognition system: flesh toning, segmentation, face detection, and face identification. Mathew et al. [45] 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.

2.1.2 Ray Tracing

The use of ray packets to exploit SIMD processing was first introduced by Wald et al. [80]. The original implementation targets the x86 SSE extensions, which execute operations using a SIMD width of four, and consequently uses packets of four rays. Later implementations use larger packet sizes of $4 \times 4$ rays [5], but these fixed-size packets are neither split nor reordered. Reshetov [68] has shown that even for narrow SIMD units, perfectly specular reflection rays undergoing multiple
bounces quickly lead to almost completely incoherent ray packets and \( \frac{1}{N} \) SIMD efficiency. Thus, worst-case SIMD efficiency is not only a theoretical possibility, but has been demonstrated in current packet-based ray tracing algorithms. Stream filtering in CoGenE maintains high efficiency when processing seemingly incoherent groups of rays, including the secondary rays required for a number of important visual effects. Evaluation in chapter ?? demonstrates that it is possible to achieve high SIMD utilization for wide SIMD units by employing filtering to remove the inactive subsets.

Several recent works have investigated the problem of coherence in secondary rays. Boulos et al. [8] describe packet assembly techniques that achieve similar performance (in terms of rays/second) for distribution ray tracing as for standard recursive ray tracing. Similarly, Mansson et al. [44] describe several coherence metrics for ray reordering to achieve interactive performance with secondary rays. Instead of tracing rays in a depth-first manner, several works have investigated breadth-first ray traversal. Nakamaru and Ohno [51] describe one such algorithm designed to minimize accesses to scene data and maximize the number of rays processed at a time. Mahovsky and Wyvill [43] have explored breadth-first traversal of bounding volume hierarchies (BVHs) to render complex models with progressively compressed BVHs. This approach, however, uses breadth-first traversal to amortize decompression cost and does not target either interactive performance or SIMD processing. CoGenE builds on these ideas to extract maximum coherence in arbitrarily-sized groups of rays.

2.2 Compilers and Scheduling

Improving performance or power via VLIW techniques is a common theme in modern embedded systems [3] including mapping and instruction scheduling techniques [41, 75]. However, these efforts do not address low-level communication issues. CALiBeR reduces memory pressure in VLIW systems but cannot directly schedule activities to reduce register file communication at the cluster level [2]. Tiwari et al. have explored scheduling algorithms for less flexible architectures
which split an application between a general purpose processor and an ASIC [77]. Eckstein and Krall focus on minimizing the cost of local variable access to reduce power consumption in DSP processors [21].

Park et al. [55] discuss a graph based software pipelining technique for mapping loops on coarse grain reconfigurable architectures. They have shown performance optimization sacrifices several opportunities for energy reduction. They stress the need for compilation techniques that optimize energy consumption, and employ techniques that significantly reduce energy consumption while minimally degrading performance. High-performance compilation techniques have also been investigated: RAW [40], CGRAs [55], Imagine [69], and Merrimac [19]. 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 motivates CoGenE’s fine-grained resource control approach. The main difference is that CoGenE’s methodology 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 [15, 84]. A similar approach by Pozzi also provides graph-based optimizations for micro-architectural constraints such as limited register ports [57]. The main differences between these efforts and CoGenE is that our DSA model is more autonomous and attempts to co-optimize performance and energy consumption rather than just performance. Scheduling techniques for power-efficient embedded processors have achieved reasonably low power operation but they have not achieved the energy-delay efficiency of our architecture [30].

2.3 Embedded Architectures

Recent approaches [12, 49, 55] have proposed the design of programmable processors or coarse grained reconfigurable arrays for video processing or wireless algorithms. These devices work in various modes to alternatively execute sequential code and the parallel kernels. The problem is that sequential and parallel codes
exhibit different kinds of parallelism and their execution time varies across different kernels within a domain. For rapidly evolving applications with stringent real time requirements, these devices will be inefficient at extracting different kinds of parallelism and may incur frequent mode changes, thereby degrading application performance. Compiler directed approaches that are similar to our effort [50] target FPGA devices as opposed to the custom silicon targeted in CoGenE.

The MOVE family of architectures explored the concept of transport triggering where computation is done by transferring values to the operand registers of a function unit and starting an operation implicitly via a move targeting a trigger register associated with the function unit [31]. In this dissertation, this concept is used for data transfer between function units.

Application specific clusters are investigated in [39, 22]. These complementary scheduler approaches minimize inter- rather than intra-cluster communication and therefore are not able to optimize register utilization as described in this work. In some sense, the fine grain horizontal microcode approach taken here can be viewed as a fine-grained extension of the VLIW concept. However the addition of more sophisticated address generators, multiple address contexts per address generator, the removal of the register file, and the fine-grained steering of data are aspects of this work that are not evident in these other efforts.

The other parallelism approach that is becoming increasingly popular is short vector or SIMD data parallelism [52, 9]. These techniques have been shown to improve performance by up to an order of magnitude on DSP-style algorithms and even on some small speech processing codes [35]. CoGenE is capable of capitalizing on this form of data parallelism as well. From an energy-delay perspective, however, it was found that SIMD operation [32] does not generally have an advantage. Tensilica’s Xtensa system [24], 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. In contrast, our ISA-independent approach mimics the data flow within the application.
closely and significantly reduces the control and access overhead. CoGenE is driven by an application suite and our architecture provides a richer set of options than a traditional more coarse grained VLIW approach.

2.3.1 Architectural Support for Ray Tracing

Packet-based ray tracing has also been exploited successfully in special-purpose ray tracing hardware projects [72, 82]. We generalize packet-based ray tracing to process arbitrarily sized groups of rays efficiently in wide SIMD environments. While commercial implementations like the G80 [53] and the R770 [4] provide wider-than-four SIMD capability, these machines employ the execution core for address computations and hence, interfere and compete with the actual data computations for resources, thus degrading performance. The Larrabee project [73] employs a many-core task-parallel architecture to support a variety of applications. In contrast, StreamRay extracts performance from ray tracing by efficiently isolating the core tasks of stream generation and stream processing to deliver high performance.

2.3.2 Design Space Exploration

Recent research has investigated exploration techniques [37, 1, 27, 84, 74] to automate the design of application specialized processors or accelerators. Based on the type of architectures explored, these techniques can be classified into three relevant categories. First, [37, 74] have investigated analytical techniques for the automation of super-scalar processors for SPEC or media application kernels. While [37] is fast, the algorithm was evaluated for one particular program phase, rather than all computational intensive components. [48, 61] have shown that complex multimedia applications like face and speech recognition consists of multiple compute intensive phases. Many different algorithms could be employed to perform the same kernel functions. While Silvano et al. [74] address this issue, their architectural analysis focuses on the memory system and not in great detail on the interconnect and execution units.

The second class of architectures explored for automation is transparent accelerators [84] for embedded systems. This study balances compilation and accelerator
constraints and is similar to our approach. While their approach is based on instruction set customization, ours is tailored to extract the data flow patterns within the application. The third and final class of architectures, including our study, fall into the category of long word machines. The PICO design system [1] consists of a VLIW GPP and an optional non-programmable accelerator and tries to identify a cost effective combination of the two. Our approach explores the design of a programmable DSA that satisfies the energy-performance-area constraints for the entire application domain.

Grovels et al. [17] employed the idea of lost cycles analysis for predicting the source of overheads in a parallel program. They present a tool to analyze performance trade-offs among parallel implementations for a 2D FFT. The CoGenE design explorer explores many design points in the architecture space for diverse application domains. Other studies [33] have explored machine learning based modeling for design spaces and this could potentially replace the simulator employed in our study. In contrast, CoGenE automatically searches the design space to arrive at optimal design points for varying constraints.
CHAPTER 3
FROM APPLICATIONS TO ARCHITECTURE

The effectiveness of the framework is evaluated for four different application domains: face recognition, speech recognition, wireless telephony, and ray tracing. The source code for the workloads were obtained from application software research groups in various universities [18, 16, 32, 48]. The applications were manually factored into sequential code and kernels. Each of the C-based kernels were then modified to fit the stream processing model required as input to CoGenE. Mathew et al. [48] performed a complete characterization of the speech recognition domain and contributed to the initial architectural methodology. Ibrahim et al. [32] characterized the wireless telephony domain. In both cases, manual effort was involved in generating object code for execution on the architecture. This dissertation performs a detailed characterization of the face recognition and the ray tracing domain. In addition, this dissertation presents the design of the optimizing compiler and the explorer that automatically designs the DSA. This chapter begins with a brief overview of speech recognition and wireless telephony. A detailed characterization of the the face recognition domain is then performed to illustrate the salient features of the architectural methodology. The complete process incurs man-months of design time and serves to motivate the need for the CoGenE automation toolkit.

3.1 Speech Recognition Overview

The speech recognition application consists of three phases that contribute to 99% of total execution time: preprocessing, HMM, and the GAU phase [46]. Preprocessing converts the raw input signal into feature vectors and is dominated by floating point computations. Nevertheless, it contributes to only 1% of the total
execution time. GAU and HMM represent Gaussian probability density evaluation and hidden Markov model evaluation respectively. GAU occupies 57.5% and HMM consumes 41.5% of the execution time of the Sphinx 3.2 speech recognition system. Both Gaussian distributions and hidden Markov models are components of most mature speech recognizers [38, 85]. GAU computes how closely a 10 ms frame of speech matches a known Gaussian probability distribution. One input packet corresponds to evaluating a single acoustic model state over 10 frames of a speech signal. A real-time recognizer needs to process 600,000 invocations of the GAU algorithm every second. The HMM algorithm performs a Viterbi search over a hidden Markov model corresponding to one model state. One input packet to the HMM implementation consists of 32 five-state hidden Markov models. While the GAU algorithm is entirely floating point, the HMM algorithm is dominated by integer compare and select operations. Its average rate of invocation varies significantly with context, but to guarantee real-time performance it is assumed in this research that all HMM models are evaluated thereby brute forcing a large component of speech processing.

3.2 Wireless Telephony Overview

Due to the existence of many different wireless communication protocols [59], the most important kernels from signal processing and wireless communication domains are chosen to form a benchmark suite. The matrix multiply, vec_max, and the dotp_sqr kernels are chosen from the signal processing domain. While vec_max selects the maximum amongst a 128 element vector, the dot products V1.V1 and V1.V2 of two input vectors V1 and V2 is computed in dotp_sqr. The other three applications were selected from the new 3G wireless telephony standard [32]. T-FIR is a 16-tap transpose FIR filter. The Rake receiver extracts signals from multi-path aliasing effects and the implementation involves four complex correlation fingers. Turbo decoder is a complex encoding application that exhibits superior error correction capabilities. This implementation contains 2 max-log-MAP modules, an interleaver, and a de-interleaver.
3.3 Face Recognition System Overview

The human face recognition problem is a complex task given the diverse range of facial features and skin tone variations. The importance of face recognition has motivated numerous algorithms [56, 14] and recognition accuracy evaluation efforts [56]. Face recognition can be viewed as two sequential phases: 1) face detection which analyzes video or camera frames to produce a set of normalized skin-tone patches which likely contain a face, and 2) face identification which compares the patches to a database of target faces to determine a probable match. Some of the face detection techniques are essentially generalized methods of object detection, and can be adapted to perform other visual feature recognition tasks.

For embedded systems, there is a natural bias towards using cheap, low-resolution cameras. Images may be poorly lit, contain occlusions, and may not contain frontal views. Figure 3.1 shows the major steps involved in face recognition. The input to the 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 state is maintained to perform motion tracking. The process is a pipeline of kernels, and the goal is to process them in real time. Flesh toning looks for patches of skin toned pixels. Segmentation looks for a patch that is big enough to contain a face and performs edge smoothing to create a patch. To facilitate processing by the next stage, the patch is output in a rectangular manner. Face detection looks for features in the patch which correspond to facial features such as eyes, ears, nose, etc. Eye location pinpoints the probable eye location candidates and normalizes the patch to meet the Face Recognition Technology (FERET) [56] normalization requirements. It also creates a boundary description for the patch. Face recognition 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 recognition algorithms and has evaluated their accuracy [18, 16]. Two algorithms (PCA/LDA and EBGM) were chosen 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 and together represent a diverse domain. The study in [48] employs the PCA technique. A brief description of the different components in a complete face recognition system is followed by a study of the execution profile of the system and its memory requirements. These techniques are also useful in general visual feature and gesture recognition systems.

3.3.1 Preprocessing: Flesh Toning and Segmentation

Skin colors are more tightly clustered in the HSV (Hue, Saturation, Value) or the NCC (Normalized Color Co-ordinated) color space than in the normally employed RGB encoding space. Pixels are thus converted from RGB space to the HSV color space and the NCC space. To improve accuracy, the consensus of two separate flesh toning algorithms based on the NCC and the HSV color spaces are employed respectively [45, 6, 76]. 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 face detection mechanism requires rectangular regions for its operation, two simple mathematical operators are performed: erosion and dilation. An erosion operator examines each pixel and blacks it out unless all its neighbors in a 3x3 pixel map are set [29]. This makes sure that small occlusions are cut away. Dilation then lights up the pixel if any of its neighbors in a 4x4 window are set.

3.3.2 Viola-Jones Face Detection

The face detector phase is based on the Viola-Jones approach which is similar in purpose to the AdaBoost algorithm [71, 78]. 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. The 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 [70]. 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). 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. A simple voting mechanism decides the final detection locations. In this approach, a detector with 100 different matching criteria is employed.

3.3.3 Holistic Face Recognition: PCA+LDA algorithm

Our PCA based face recognition algorithm is based on [83]. This algorithm was preferred over the Eigenfaces technique [45] due to the increased recognition accuracy in the original FERET study. In the first step, 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.

3.3.4 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 [81]. 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. A re-implementation of the EBGM algorithm was provided by the CSU research group [7]. 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 in order 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 in the algorithm 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, the version in this dissertation employs sufficient code motion and reordering to process the image information on a real-time frame-rate basis.

### 3.4 Workload Characterization

Figure 3.2 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.

### 3.4.1 Memory Characteristics

Memory and execution characteristics studies are based on the SimpleScalar [11] simulation framework with architectural parameters chosen to model an out of order processor (1.7 GHz) similar to a Alpha 21264. The test configuration is a baseline machine with four integer and four floating point units each in order to provide sufficient execution resources, a 2MB L2 cache, and a 600 MHz DRAM interface. In addition, the size of the caches, the number of integer units, and the number of floating point units are varied for sensitivity analysis.

Figure 3.3 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 3.4 shows the L2 cache (unified) hit rates for the same configurations. The L2 hit rates are computed 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.

3.4.2 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 3.1 shows the instructions committed per cycle (IPC) for four different configurations as the number of integer and floating point function units vary. It can be observed that adding more functional units does not provide a commensurate increase in performance. The configuration with 4 integer and 4 floating point units outperforms the one with 2+2 units by a marginal 5%. In addition, performance saturates beyond six units (3+3). Table 3.2 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
Input Stream

Flesh Toning → Segmentation → Face Detection → Eye Location

Face Identified ← Face Recognition

**Figure 3.1.** Processing Kernels in a Face Recognition System

**Figure 3.2.** Execution profile for PCA/LDA and EBGM face recognition systems

**Figure 3.3.** L1 cache miss rates
application is at least 2 times slower than real time with 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. They are summarized below:

- The face recognition kernels commonly perform a lot of computations of the form $Z[i] = Z[i-1] + \sum_{j=0}^{m} X[j] \times Y[W[j]]$ which contains loop carried dependencies.

- The problem is further exacerbated in multi-level loops where such computations entail complex indirect accesses.

- 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.

- 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.

### 3.5 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. Three distributed 8KB SRAMs
Figure 3.4. L2 cache hit rates

<table>
<thead>
<tr>
<th>Num. XUs</th>
<th>PCA/LDA complete</th>
<th>EBGM complete</th>
<th>PCA/LDA alone</th>
<th>EBGM alone</th>
</tr>
</thead>
<tbody>
<tr>
<td>1+1</td>
<td>0.651</td>
<td>0.623</td>
<td>0.780</td>
<td>0.757</td>
</tr>
<tr>
<td>2+2</td>
<td>0.703</td>
<td>0.683</td>
<td>0.830</td>
<td>0.793</td>
</tr>
<tr>
<td>3+3</td>
<td>0.727</td>
<td>0.712</td>
<td>0.897</td>
<td>0.877</td>
</tr>
<tr>
<td>4+4</td>
<td>0.729</td>
<td>0.720</td>
<td>0.905</td>
<td>0.890</td>
</tr>
</tbody>
</table>

Table 3.1. Instructions per Cycle (IPC) for baseline alpha configuration with varying number of execution units (XUs)

<table>
<thead>
<tr>
<th>Num. XUs</th>
<th>PCA/LDA complete</th>
<th>EBGM complete</th>
<th>PCA/LDA alone</th>
<th>EBGM alone</th>
</tr>
</thead>
<tbody>
<tr>
<td>1+1</td>
<td>2.310</td>
<td>2.560</td>
<td>1.530</td>
<td>1.610</td>
</tr>
<tr>
<td>2+2</td>
<td>2.050</td>
<td>2.107</td>
<td>1.378</td>
<td>1.383</td>
</tr>
<tr>
<td>3+3</td>
<td>1.800</td>
<td>1.870</td>
<td>1.040</td>
<td>1.160</td>
</tr>
<tr>
<td>4+4</td>
<td>1.780</td>
<td>1.784</td>
<td>0.978</td>
<td>1.003</td>
</tr>
</tbody>
</table>

Table 3.2. Speedup/slowdown over real time corresponding to 5 frames per second (real time is scaled to 1)
(input, output, and scratch) were employed for the face recognition 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 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.

3.5.1 DSA memory Architecture

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. A hardware loop unit (HLU) is 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 [48]. The Viola/Jones detection kernel requires a maximum of three simultaneous loop contexts. Hence, the 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.

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. Overall, the memory system for our DSA consists of a loop unit, three distributed 8KB SRAMs with two ports each, and associated AGUs.
3.5.2 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 one computation of work in the function unit. 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.

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 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 face recognition.
CHAPTER 4

DSA SYSTEM ARCHITECTURE

At the system level, this dissertation employs a heterogeneous multiprocessor and comprises a general purpose processor (GPP) for sequential code and a DSA to accelerate the kernels. The architecture, depicted in Figure 4.1, is an example of a decoupled access-execute architecture [54]. The host GPP handles general control and set-up duties and moves data to and from the DSA via double buffered input and output SRAMs.

The DSA is shown in Figure 4.2. The memory system includes a HLU, dual-ported and double buffered SRAMs, and address generator units (AGU). Each HLU context stores the current value of the loop variables in a kernel’s loop nest. If multiple kernels will be concurrently active, then multiple contexts are necessary to avoid delays in reloading context data into the HLU. The loop variable values are used by the AGU’s for generating addresses to support sophisticated addressing modes [47]. The use of multiple SRAMs provides higher memory bandwidth. Each SRAM is role specific in this stream based DSA strategy, in which applications consume input frames to produce output data and state information for subsequent frame processing. Since, the input SRAM is double buffered, the host processor loads the next input frame while the DSA is processing the current frame. The output SRAM is similarly structured so the host processor can remove the previous frame outputs while the DSA is generating the current frame outputs. The scratch SRAM may be dual ported, but in this case both ports would be used by the DSA in order to increase state data bandwidth. The HLU permits modulo scheduling [67] of loops whose loop counts are not known at compile time and this capability reduces compilation complexity.
Figure 4.1. Heterogeneous Multiprocessor Organization

Figure 4.2. Organization of the recognition DSA
The horizontal microcode approach allows the multiplexer based interconnect to be configured under program control. This allows function units and their associated pipeline registers to be linked to create pipelines which persist for as long as they are needed. This persistent pipeline characteristic is similar to the fixed yet inflexible pipelines found in application specific integrated circuits (ASICs) and is a significant factor in the energy-delay efficiency of the approach. Value lifetime and motion are also under program control. The compiler generated microcode controls data steering, clock gating (including pipeline registers), function unit utilization, and single-cycle reconfiguration of the address generators associated with the SRAM ports. A functional unit can either be an integer or floating point execution unit or a register file. As with any highly parallel system, the interconnect subsystem is performance critical. Operating frequency can be increased by reducing individual multiplexor widths and/or adding additional multiplexor levels. The result is improved interconnect throughput at the cost of a slight increase in fall-through delay.

### 4.1 DSA Evaluation for Face Recognition

Figure 4.4 compares the IPC of the baseline alpha machine with different DSA configurations: i) DSA with perfect back-end implies no stalls due to communication or execution resources, which shows the performance of the memory system, ii) DSA with perfect memory system, which indicates the performance of the interconnect and execution cluster back-end, iii) Complete DSA configuration with actual memory and back-end, but with seven functional units and the register file, and iv) Complete DSA configuration with eight functional units and no register file. It can be observed that the DSA 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 configuration with perfect memory evaluates the cluster back-end. When
compared to the Alpha processor, this configuration provides a 3x improvement for face detection and 6.7x improvement for face identification. The advantage comes from exploiting persistent pipeline 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.

The last two configurations in figure 4.4 show the performance of the complete 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 complete DSA provides as much as a 2.7x performance improvement for face detection and a 5.5x-5.8x improvement 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.

Comparing the complete 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 fine grained horizontal microcoded nature of the DSA implies that the
Figure 4.3. Functional Unit Architecture

Figure 4.4. Plots showing the potential for memory parallelism and 'ASIC-like' flows
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. The CoGenE compiler, that alleviates code generation time is described in chapter ??.

This is followed by a discussion of the CoGenE design space explorer in chapter ??.
REFERENCES


[58] Press, T. A. Techies dissect apple’s iphone.


