Thesis proposal
CoGenE: Automating the Design of Embedded Domain Specific Architectures

Karthik Ramani,
Advisor: Dr. Al Davis
School of Computing
University of Utah

Abstract

Future embedded processors are likely to integrate heterogeneous cores on a single die to increase compute and power efficiency. These processors are likely to contain a combination of general purpose processors (GPPs) and specialized accelerators (DSAs). Design and fabrication costs for these processors are significant and conflict with the dynamic nature of embedded applications, and the short time-to-market requirements for embedded systems. An ideal solution to minimizing design time and costs is to totally automate the design process. The application expert can employ this automatic tool to take an application suite and constraints on device cost, power, and performance as inputs and produce a validated design (GDSII file) that is ready for fabrication. This requires that we automate the three major steps: i. Code splitting: decomposing the application program into sequential code that runs on a GPP, and parallel code, that runs on the specialized accelerator ii. Accelerator design: designing an architecture that delivers the performance and power requirements under different constraints (short time to market, very high performance, minimum area, etc.), and, iii. Kernel compilation: mapping the parallel code into the specialized architecture in an easy and efficient manner. This thesis addresses the latter two problems and proposes a compilation framework that automates the design of domain specific architectures for embedded systems.

In the first part of the thesis, we showcase an architectural design methodology that generates a specialized core with high performance and low power characteristics. Specifically, we create ‘ASIC-like’ execution flows to sustain the high memory parallelism generated within the core. This benefit comes at a significant increase in programming complexity and time, and necessitates co-scheduling the often conflicting constraints of data access, data movement, and computation. To address this problem, we propose a modular compiler approach that employs integer linear programming (ILP) based ‘interconnect-aware’ scheduling techniques for automatic code generation. This removes the need for man-months of manual scheduling and minimizes design time.

Stall Cycle Analysis (SCA) is an iterative technique which exploits the fact that execution cycles of a program consists of pure computation and stall cycles which occur due to the presence of bottlenecks. Compilation and architectural simulation can be employed to analyze these bottlenecks and provide insight into architectural avenues for further performance/energy improvements. In the second part of the thesis, We create an SCA based exploration tool that intelligently prunes the design space from ≥1000 points to tens of points for three application domains: face recognition, speech recognition, and wireless telephony. To our knowledge, this is the first study that leverages compilation and architectural simulation to automate the design of compiler friendly DSAs while significantly reducing design time from months to hours.
1 Introduction

1.1 Problem Description

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 compute, power, and energy efficiency has motivated the trend towards heterogeneous processors on a single die (e.g. AMD Fusion, Intel Larrabee, TI OMAP). Information fusion, where one device is required to execute a plethora of applications from variegated market segments like image processing, data retrieval, and signal processing, etc., has further exacerbated the problem. These applications are generally characterized by inter-twined sequential and parallel code (called kernels) phases. Simultaneously, the rapidly growing deployment of embedded computing devices and the dynamics of the competitive market dictate very short design cycles.

Creating embedded devices that adapt and cater to the dynamic needs of the application requires that we optimize their design at multiple levels: application, operating system design, compilation, architecture and circuit design, and during fabrication. This study optimizes the design of embedded computing devices by leveraging compilation and architectural design. Specifically, we propose a design framework that leverages compilation and architectural design in a symbiotic manner to automate the design of performance-energy optimal processors.

1.2 Background

Traditionally, general purpose processors (GPPs) were used to execute sequential setup/control code, while throughput-efficient application specific integrated circuits (ASICs) were employed for compute intensive kernels. 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).

Recent approaches [1, 6] 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.

This thesis begins with an architectural design style that provides 'ASIC-like’ data flows to deliver the performance requirements of real time applications in embedded power budgets. Setting up these high throughput pipelines requires co-scheduling of multiple conflicting constraints. A modular compiler approach that employs novel interconnect scheduling algorithms performs co-scheduling of these constraints. While this approach is successful in designing real time embedded devices, it incurs a significant time for a system designer to understand the application requirements and then arrive at a design. Since this conflicts with the short time-to-market requirements for embedded systems, we propose an automatic design space exploration tool. The tool employs an idea called as ‘Stall Cycle Analysis’ to prune the design space in an intelligent manner and arrives at performance-energy optimal designs for embedded DSAs. Thus, design time is reduced from man-months of workload characterization to hours of automatic exploration.
This thesis proposal is organized in two parts. In the first part, we discuss all completed work briefly. We then discuss current and proposed work (Section 2) along with a tentative time-line.

1.3 Completed Work: Step I

In this section, we use the face recognition domain as a case study to describe our architectural design methodology. Other work [5, 2] has demonstrated the effectiveness of a similar approach for the speech recognition and the wireless telephony domains. During design space exploration, we will further demonstrate the effectiveness of our approach for three application domains in the embedded space: speech recognition, face recognition, and wireless telephony. In addition, we are also investigating the generality of the CoGenE approach for the high performance ray tracing application 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 of the compute, control, and data access characteristics of all the kernels is performed to create the ArcFace 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.

Architecture Description

The memory architecture of the DSA [8] 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 (Instructions per cycle) 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 [10]. 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 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 [5, 2] 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.

The face recognition DSA

The face recognition DSA (ArcFace), is shown in figure 2. 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 3. 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.

**Compilation Methodology**  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, 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. The overall back-end compilation
flow is illustrated in 4. After preliminary control and data flow analysis is performed, we identify the inner most loop and load the initiation interval into the hardware loop unit (HLU) and it provides support for modulo scheduling. After register assignment, we perform Integer Linear Programming (ILP) based interconnection scheduling followed by post pass scheduling to resolve conflicts that may arise due to ILP scheduling.

**Benchmarks** We evaluated our design using codes that cover key aspects of face recognition. These include flesh-tone and color coordinate conversion, image segmentation, face detection, and eye-location, and two face identification benchmarks (EBGM and PCA/LDA algorithms). The FIR benchmark was included from the signal processing domain to test the generality of our approach.

**Results Summary** Initial results show that the compiler can deliver 1.65x the throughput of a high performance Pentium 4 processor while providing 10.65x energy savings than a low power XScale processor. Overall, the results demonstrate that face recognition can be performed in real time in an embedded power budget.

**1.4 Completed Work: Step II**

In the above work, we observed that designing a DSA for an application domain incurs significant time in three different sub-processes: workload characterization, architecture design and simultaneous compilation flow design. The total time for the complete process conflicts with the competitive nature of the embedded market and hence, we need an automatic tool that explores the architectural design space and produces a performance-energy optimal architecture. We discuss the final step of design automation now.
This thesis explores a novel exploration algorithm [7] to automate the design of performance-energy optimal DSAs for constraints like minimum area, maximum performance, etc. This iterative algorithm is termed 'Stall Cycle Analysis' (SCA) and is based on the observation that total execution cycles of a program consists of cycles used for pure computation and stall cycles that occur due to the presence of performance/energy bottlenecks. Understanding these bottlenecks can provide us with insight for further performance improvements (energy reduction). This observation is applied during compilation and simulation and is used to guide the exploration process for designing optimal design points in the architectural phase. As opposed to other studies [3], the design automation process also guarantees optimized compilation and to our knowledge, this is the first study that employs compilation and architectural simulation in a symbiotic manner for design space exploration. In summary, the contributions of the study are:

SCA is employed in two phases on a given test architecture. Bottleneck diagnosis (BD) associates stall cycles in the compilation schedule (or simulator) to bottlenecks for performance (energy). Examples of such bottlenecks are insufficient parallelism in hardware, resource contention, resource starvation, routability issues, etc. During diagnosis, we classify and analyze bottlenecks to investigate various architectural solutions. This output of this phase can be used by the system designer in making further architectural choices. In our framework, this information is fed as input to the design selection (DSel) phase. Those solutions that can potentially lead to improved performance (reduced energy) are identified and we add (Dilation) or remove (Thinning) resources to generate the next test architecture. This process is repeated in an iterative manner to arrive at performance-energy optimal designs for different constraints.

**Performance and Power Estimation** The effect of compilation can be analyzed with the cycle-accurate simulator and it estimates power using high level parameterizable power models. Our power models [9] (90 nm node) employ analytical models from for all predictable structures and empirical models similar to for complex structures like the hardware loop unit (HLU). Our area estimates are obtained from Synopsys MCL and design compiler scaled to 90 nm.

**Benchmarks** Our benchmarks consists of seven kernels from face recognition, three kernels from speech recognition, and six kernels from wireless telephony domains. The speech recognition application consists of three phases that contribute to 99% of total execution time: preprocessing, HMM, and the Gaussian phase [4]. The kernels from the wireless domain include predominant operations like matrix multiplication, dot product evaluation, determining maximum element in a vector, decoding operations like rake and turbo, and the FIR application.

**Results summary** As compared to manual designs optimized for a particular metric, SCA automates the design of performance-energy optimal DSAs for minimum energy-delay product (17% improvement for wireless telephony), minimum area (75% smaller design for face recognition), or maximum performance (38% improvement for speech recognition). To our knowledge, this is the first study that leverages compilation and architectural simulation to design compiler friendly DSAs, thereby significantly reducing DSA design time (months to hours).

**Preliminary Conclusions** The major contributions of this study are:

- We employ face recognition as a characterization case study to demonstrate the effectiveness of our architectural design methodology.
- We present the CoGenE compiler framework that employs ILP based interconnect scheduling to map the applications (all three domains) to the DSA, and this provides for automatic code generation.
• Stall Cycle Analysis (SCA), a robust exploration algorithm that efficiently combines optimized compilation and architectural simulation for the design of energy efficient soft real time embedded systems.

• Design case studies that employ SCA to automate the design of performance-energy optimal DSAs for minimum energy-delay product, minimum area, and maximum performance. As compared to time-consuming manual design, SCA provides a richer set of choices to the designer while reducing design time significantly (hours on an 1.6 GHz AMD Athlon PC).

• The generality of the CoGenE approach is demonstrated with the help of DSA designs for three different application domains in the embedded space: speech recognition, face recognition, and wireless telephony.

• As part of current (and proposed) work, the applicability of CoGenE is also being investigated for the high performance ray tracing domain. This will further enhance the robustness of the infrastructure.

Thus, we now have an automated design framework that can analyze a variety of application domains and design compiler friendly DSAs in a very short time.

2 Further Work

2.1 Compilation for Speech Recognition and Wireless Telephony

The exploration phase demonstrates the effectiveness of the compiler in generating code for all three application domains. Further investigation is required to improve the efficiency of compilation. In specific, we would like to approach the performance of hand scheduling for various domains. This would increase the robustness of the CoGenE infrastructure and help the application designer in arriving at optimal designs at a faster rate.

Until now, the utility of the CoGenE infrastructure was tested for three application domains in the embedded space. We would like to investigate the generality of our approach further by investigating CoGenE for the high performance ray tracing domain.

2.2 Extension to the Ray Tracing Application Domain

At present every computer has a dedicated processor that enabled 3D graphics. These graphics processing units implement the z-buffer algorithm and can interactively display several million triangles with image based textures and lighting. While the wide availability of GPUs has revolutionized the graphics industry, there are some classes of applications that have not significantly benefited from these devices. Examples of such applications are vehicle design, landscape design, manufacturing, etc., and these programs have huge datasets with non-polygonal data and require high quality shadows, reflection, and refraction effects. These classes of applications typically benefit from an algorithm called ray tracing.

Ray tracing is a global illumination based rendering point-sampling technique that traces light rays from the eye back through the image place into the scene. Current processors do not have the compute power to perform a huge number of floating point operations to implement real time ray tracing. Moreover, the complex memory access patterns exhibited by ray tracing necessitates understanding the different phases in the algorithm. To further complicate architectural design, there seems to be a huge number of different implementations of the algorithm. There seems to be little/no agreement in literature as to what constitutes a ray tracing suite that architects should target. Designing an architecture for ray tracing thus necessitates the following steps:
Ray trace benchmark suite, establishment of a suite that contains a diverse implementation of different ray tracers. This suite would represent combine the salient features of all ray tracers with respect to visual perception, application target, and computational complexity. Architects can target their designs to deliver good performance for this generic benchmark suite.

Characterization, Ray tracers can be generally characterized by the following three kernels: i) ray-acceleration structure intersection, intersecting a ray with the acceleration structure that encapsulates the scene objects, ii) ray-primitive indisposition, intersecting the ray with primitive objects contained in the acceleration structure that is hit, and iii) shading, computing the color and the illumination of the pixel. Understanding the memory, control, and computational needs of the different ray tracers will help us understand the architectural implications of such applications.

Hardware for ray tracing, Improving the existing architecture that delivers the performance requirements of the ray trace benchmark suite. Initial studies have demonstrated that a multi-threaded architecture is a good match for ray tracing. The CoGenE approach can be employed to accelerate the parallel phases with the architecture.

This study will focus on the establishment of the ray trace benchmark suite and in improving the performance of the multi-threaded architectural approach. In addition, improving the current infrastructure to address some of our current shortcomings (instruction scheduling, architectural improvements, etc.) is also part of proposed work.

2.3 Tentative Time-line

A time line for the proposed work is shown below:

- December 2007: Discussion to acquire various ray trace benchmarks and scenes that architects need to target. Table 2 shows the various ray tracers that are under consideration. The benchmarks to be targeted will be finalized after further discussions.
- February 2008: Begin characterization of the ray tracing benchmark suite. Evaluate compilation for wireless and speech recognition domains.

A complete time-line for the duration of study is shown in table 1.

2.4 Future Work

Future billion transistor architectures will have a multitude of cores on a single die. These cores will vary in size, functionality, and in power dissipation and have to work in a symbiotic manner to deliver high performance for the target application. At the same time, newer application domains will continue to arrive.
<table>
<thead>
<tr>
<th>Semester</th>
<th>Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fall 2004</td>
<td>Workload characterization of face recognition</td>
</tr>
<tr>
<td></td>
<td>Investigation into heterogeneous interconnects (WCED 2004)</td>
</tr>
<tr>
<td>Spring 2005</td>
<td>DSA design for face recognition</td>
</tr>
<tr>
<td></td>
<td>Investigation into heterogeneous interconnects (HPCA 05)</td>
</tr>
<tr>
<td>Summer 2005</td>
<td>Investigation into compilation infrastructure</td>
</tr>
<tr>
<td></td>
<td>Power modeling for clustered processors and CMPs</td>
</tr>
<tr>
<td>Fall 2005</td>
<td>Infrastructure modifications: Trimaran to CoGenE</td>
</tr>
<tr>
<td></td>
<td>Power efficiency exploration in clustered processors (ISPASS 06)</td>
</tr>
<tr>
<td></td>
<td>Accelerating coherence protocols (ISCA 06, IEEE Micro 2007)</td>
</tr>
<tr>
<td>Spring 2006</td>
<td>Internship at FACT, Intel</td>
</tr>
<tr>
<td></td>
<td>Platform acceleration for future Intel processors</td>
</tr>
<tr>
<td>Summer 2006</td>
<td>Investigation of compilation techniques for face recognition</td>
</tr>
<tr>
<td>Fall 2006</td>
<td>A compiler friendly DSA for face recognition (CASES 2007)</td>
</tr>
<tr>
<td></td>
<td>Investigation of exploration algorithms</td>
</tr>
<tr>
<td>Spring 2007</td>
<td>Compilation for speech recognition</td>
</tr>
<tr>
<td></td>
<td>SCA investigation for wireless algorithms</td>
</tr>
<tr>
<td></td>
<td>Initial discussions for design space pruning</td>
</tr>
<tr>
<td>Summer 2007</td>
<td>Internship at ATI Research</td>
</tr>
<tr>
<td></td>
<td>Power Modeling for dynamic market requirements</td>
</tr>
<tr>
<td></td>
<td>Power Efficiency Exploration for future GPUs (GPGPU 07)</td>
</tr>
<tr>
<td>Fall 2007</td>
<td>Improvements to SCA for three applications domains</td>
</tr>
<tr>
<td></td>
<td>Improvements to CoGenE (submitted to ISCA 2008)</td>
</tr>
<tr>
<td></td>
<td>Initial Discussions for ray tracing</td>
</tr>
<tr>
<td>Spring 2008</td>
<td>Establishment of the ray trace suite</td>
</tr>
<tr>
<td>and further</td>
<td>Analyze ray trace benchmark suite</td>
</tr>
<tr>
<td></td>
<td>Compilation Infrastructure improvements</td>
</tr>
</tbody>
</table>

**Table 1. Complete research time line**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manta</td>
<td>Heavily threaded ray tracer with high speed rendering</td>
</tr>
<tr>
<td>dynBVH</td>
<td>Bounding Volume Hierarchy based ray tracer with approx. construction tech.</td>
</tr>
<tr>
<td>Galileo</td>
<td>Path tracer</td>
</tr>
<tr>
<td>Murph</td>
<td>Simple ray tracer</td>
</tr>
<tr>
<td>Whitted</td>
<td>First known global illumination model</td>
</tr>
<tr>
<td>Pavre</td>
<td>Simple C based ray tracer</td>
</tr>
<tr>
<td>Pixie</td>
<td>Open source photo-realistic renderer</td>
</tr>
</tbody>
</table>

**Table 2. Tentative ray trace benchmark suite**
Examples of such application domains are drug delivery, weather modeling, medical imaging, financial modeling, and automotive operations, etc. Given the demand for increased performance, power and thermal dissipation will continue to be a huge problem. In the face of circuit level issues, guaranteeing reliability of a many core chip will necessitate the investigation of power efficient approaches to core computing at different levels. While there are many interesting research problems to be solved, the following broad areas are interesting given our experience and inclination.

2.4.1 Emerging Application Domains

As part of future work, we intend to extend the framework to design DSAs for two application domains that are becoming increasingly important:

- **Automotive Engineering**: Vehicles that are manufactured today contain many micro-controllers and processors to perform common operations like ABS, traction control, detecting sleepiness, etc. These operations are well suited to domain specific acceleration and fit a stream processing model discussed in this study. Given the high volume and the huge application space, DSAs design for this domain is a compelling research area.

- **Finance Modeling**: Another application domain which will significantly benefit from acceleration is finance modeling. Investigating macro and micro-economic trends in various business is compute intensive and currently incurs months of computation. This manifests as a significant wastage in power requirements and in manpower. Given the importance of accelerating economic trends, this domain remains an active research area for performance improvements.

2.4.2 Core Computing

Heterogeneous computing requires us to partition applications in an intelligent manner to exploit the parallelism available in designs. Investigating semi-automatic code factoring techniques is one area of research. For example, GPUs have sufficient compute power to deliver the performance requirements of many real time applications. Leveraging factoring techniques at the program level and design techniques at the core level can improve the compute efficiency of such devices. We also intend to increase the flexibility of our compiler in investigating different algorithms for core scheduling.

References


[7] K. Ramani and A. Davis. Automating the design of embedded domain specific architectures using stall cycle analysis (sca). In *Submitted to ISCA 2008*.
