Persistent Pipelines: From Sophisticated Applications to Embedded Architectures

Abstract

Modern portable devices are expected to perform a variety of compute-intensive tasks; playing music and video, taking pictures, and running sophisticated wireless telephony and imaging software. The rate of increase in the computational demands and the number of these new applications has consistently outpaced the growth in battery capacities. Future devices will be required to deliver high performance while executing these real time applications. In addition, our desire to produce devices with high battery capacities necessitates that we dissipate very low power while performing such tasks. Traditionally, most embedded devices consist of a low power processor and a few Application Specific Integrated Circuit (ASIC) processors that execute the compute intensive kernels of some applications. While the above approach meets performance and power requirements they inherently possess many disadvantages. The rapidly increasing number of new applications and algorithmic techniques requires us to design many such fixed function devices and incurs long and expensive design cycles. General purpose processors such as DSPs solve the above problem in that they can support a variety of applications and algorithms. However, the inherent general purpose nature of the devices cannot exploit the specialized streaming nature of the applications and so, exceed the stringent power requirements of embedded devices. The first step towards achieving high performance while preserving power efficiency is to understand the nature of the computations involved in the application. After decomposing the application into a pipeline of computational kernels, we map the algorithms to a programmable VLIW architecture that possesses the best performance/power characteristics. We then present a compiler framework that generates code for the above architecture while optimizing for the energy-delay product.
The study then explores a variety of interconnection scheduling algorithms to identify the interconnect structure that best mimics the data flow in the application suite.

1 Introduction

The exponentially increasing deployment of embedded devices in appliances ranging from cellular phones to real time surveillance systems requires system engineers to design architectures targeting a diverse set of applications while possessing stringent performance and power requirements. The need for low power design while delivering the required performance necessitates that architectures closely approach the data flow inherent in the application suite. At the same time, the architecture need to be flexible enough to achieve good performance over a wide range of applications.

Traditionally, two solutions have been employed for the above problem. ASICs have been used in many embedded applications such as face recognition, speech recognition, wireless baseband receivers, etc. Wireless applications that are executed on cellular devices involve many different compute-intensive kernels that communicate with each other at a coarse granularity. The inherent real time nature of the application requires us to employ many different ASICs that execute each of the kernels while communicating with a low power processor. One of the biggest drawbacks of such an approach is that as algorithms evolve, we incur long and expensive design cycles for each of the compute kernels. Future embedded devices may require many new sophisticated applications and this may render the deployment of ASICs to be prohibitive for cost and space reasons.

Another solution that has been proposed to the problem is the use of general purpose processors (GPP). A big advantage is that a single GPP can execute all the programs in an application suite. The programming flexibility comes with the inability of the processors to identify the specialized data flow within the application. In some cases, the GPP can be tailored to deliver high performance while expending unnecessary power and this in turn, reduces the battery life of the portable device. In some cases like face recognition, the available hardware budget cannot deliver the real time performance required in the application.

Delivering the required performance at a fixed power budget for a range of applications stresses the need for a combined hardware/software co-synthesis of architectures that mimic the data flow of programs in the application suite. In this study, we make the following contributions:
• We perform a detailed characterization of a wide range of applications including speech recognition, face recognition, wireless telephony kernels, simple ray tracers, etc. and determine the memory, control flow and computational requirements of such applications.

• We map the applications to a clustered VLIW architecture with a fine grain programmable interconnect that mimics the data flow within the application and delivers significant performance improvements at very low power dissipation and thus, delivers higher battery life.

• We provide a compiler framework that generates code for each of the above applications and thus provide an architecture that delivers good performance across the application suite.

• We explore a variety of interconnect scheduling schemes and demonstrate that different schemes work well for different applications. We then propose a novel scheduling algorithm that minimizes the energy-delay product for all the applications.

• Finally, we provide a tool that delivers a compiler/architecture framework with superior performance and power characteristics for a particular application domain.

1.1 Application Suite

In this section, we give a brief overview of the different applications in our program suite. We employ a subset of programs from the Media-bench suite and some sophisticated programs that are discussed below. A detailed analysis of the programs helps in generating a template file that describes the back-end architecture.

1.1.1 Perception: Face and Speech Recognition

Recognizing speech, gestures, and visual features are important interface capabilities for future embedded devices. In our studies, we perform a detailed analysis of common perception applications including the PCA/LDA and EBGM algorithms and decompose the problem into a pipeline of processes. We then determine the complex flow requirements for each of the different algorithms and identify the algorithms that are more suitable for embedded devices.
1.1.2 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. In this study, we analyze the different execution phases of a ray tracing algorithm and quantify its compute, control flow and the memory requirements. Even though ray tracing is not a compelling application on a portable device, it strengthens our workload characterization methodology and may be suited to the data flow inherent in our architecture.

1.1.3 Wireless Telephony

Despite the rapid progress rate in embedded and DSP processing capabilities, the performance requirements of newer algorithms (3G) cannot be delivered without the deployment of ASICs. In our application suite, we select the RAKE receiver, TURBO decoder, and the TFIR programs from the 3G telephony suite. These applications have stringent storage requirements while having a fairly predictable data flow.

1.2 Persistent Pipelined VLIW architecture

Our streaming architecture delivers very high performance because of its ability to preserve maximized uninterrupted pipeline flows during the execution of the kernel. This flow is achieved with a clustered VLIW architecture that programs the inter-functional unit interconnect in a fine grain manner and thus sets up a pipeline that mimics the data flow in the application.

1.3 The Streaming Compiler framework

Our compiler infrastructure is based on the TRIMARAN compiler framework. Several significant modifications have been made to the framework in order to be able to compile for our architecture. The critical difference comes from the disparity between the traditional load-store VLIW architecture targeted by TRIMARAN and our fine-grained VLIW approach. Our architecture supports program control over communication, self-managed SRAMs with independent Address Generator Units (AGUs), clustered register files, etc.
1.3.1 Interconnect Scheduling

As discussed previously, our architecture consists of a programmable inter-functional unit interconnect that can deliver data across the system. Our previous studies have shown that programming this interconnect is critical to the performance of the system. Hence, during the code generation stage, we add a new stage after the register scheduling stage that performs the interconnect scheduling. We explore a variety of such scheduling schemes based on a combination of Integer Linear Programming (ILP) and common vision techniques and demonstrate the suitability of different techniques to different algorithms. For example, an interconnect scheduling algorithm based on popular computer vision techniques delivers the best performance for face recognition applications. Similarly, ILP techniques possess the best performance/power characteristics for wireless telephony applications. We then demonstrate a novel heuristic-based scheduling algorithm that optimizes for power and performance across all the benchmarks in the application suite.