Modeling and Implementation of Dynamic Data-Driven Application Systems

by

Kishan Sudusinghe

Research outcome report submitted to the Marshal Plan Foundation
via Salzburg University of Applied Science, in fulfillment
of the requirements for the Marshal Plan Scholarship
2014

Table of Contents

1 Introduction 1

2 Background and Related Work 5
- 2.1 Core Functional Dataflow 5
- 2.2 Topological Patterns 6
- 2.3 Related Work 7

3 Modeling Framework and Dynamic Multi-Mode Scheduling 9
- 3.1 Adaptive Classification 9
- 3.2 Modeling Methodology 10
- 3.3 Dynamic, Multi-Mode Scheduling 13

4 Case Study: Face Detection 16
- 4.1 Application Design and Experimental Setup 16
- 4.2 Experimental Results 19

5 Conclusion 23

6 Acknowledgement 24

Bibliography 25

Abstract

In this report, we investigate new design methods for data-driven digital signal processing (DSP) systems that are targeted to resource- and energy-constrained embedded environments, such as UAVs, mobile communication platforms and wireless sensor networks. Signal processing applications, such as keyword matching, speaker identification, and face recognition, are of great importance in such environments. Due to critical application constraints on energy consumption, real-time performance, computational resources, and core application accuracy, the design spaces for such applications are highly complex. Thus, conventional static methods for configuring and executing such embedded DSP systems are severely limited in the degree to which processing tasks can adapt to current operating conditions and mission requirements. We address this limitation by developing a novel design framework for multi-mode, data driven signal processing systems, where different application modes with complementary trade-offs are selected, configured, executed, and switched dynamically, in a data-driven manner. We demonstrate the utility of our proposed new design methods on an energy-constrained, multi-mode face detection application.

Chapter 1

Introduction

Embedded systems are often deployed and configured to handle multiple application tasks concurrently across different subsets of processing resources. In the domain of embedded signal processing, modern platforms consist of multiple processing cores that can concurrently support DSP- (digital signal processing)- intensive functions such as multimedia (e.g., face recognition, speaker identification, pattern recognition) and wireless communication (e.g., GSM, digital radio, NFC, Bluetooth), as well as control-oriented functions, such as those associated with user interfaces and file management (e.g., see [1]). With the increasing need for efficient and robust development of embedded systems, it is important to utilize and effectively manage the limited resources available in these computing devices dynamically in the context of data characteristics and operating conditions. Static modeling and management of execution constraints, including those involving energy consumption, real-time performance, computational resources, and core application accuracy, is not effective in designing efficient embedded systems that must adapt to time-varying requirements. Thus, in this report, we develop and demonstrate new techniques for dynamic, data-driven modeling, scheduling, monitoring, and execution of DSP applications running on resource-limited embedded platforms.

Dataflow modeling techniques are widely used to model, schedule and imple-

ment DSP systems [1]. Adaptive Stream Mining (ASM) is an important subclass of DSP applications where real-time knowledge extraction and classification are of high importance [2]. Unlike traditional data mining systems, where data is stored statically and mined through queries on the static (or slowly changing) data, ASM data arrives continuously and must be processed in real-time. Statically configured approaches to ASM processing do not scale well, with scalability problems getting worse as ASM nodes become distributed and mobile. Furthermore, integrating diverse application subsystems or diverse configurations of the same subsystem (multi-mode operation) for trade-off optimization or information integration requires adhering to global constraints on resource utilization and performance, while managing different quality-of-service characteristics of the subsystems. Therefore, novel design and implementation techniques that deviate from traditional, statically-oriented stream mining system design are needed to address the growing need for performance- and energy-optimized implementation of ASM in the context of dynamic, data-driven, and multi-mode processing scenarios. A conceptual design flow for this class of targeted multi-mode scenarios is illustrated in Figure 1.1.

This work represents a novel integration of dataflow based design methods for signal processing with the paradigm of Dynamic Data-Driven Application Systems (DDDAS). High-level, signal-processing-oriented dataflow models of computation allow designers to systematically formulate the design flow for a DSP system, and to integrate hardware, software, and application constraints into such design flows [1]. DDDAS is a paradigm that rigorously integrates application system modeling, instrumentation, and dynamic, feedback-driven adaptation of model and instrumen-

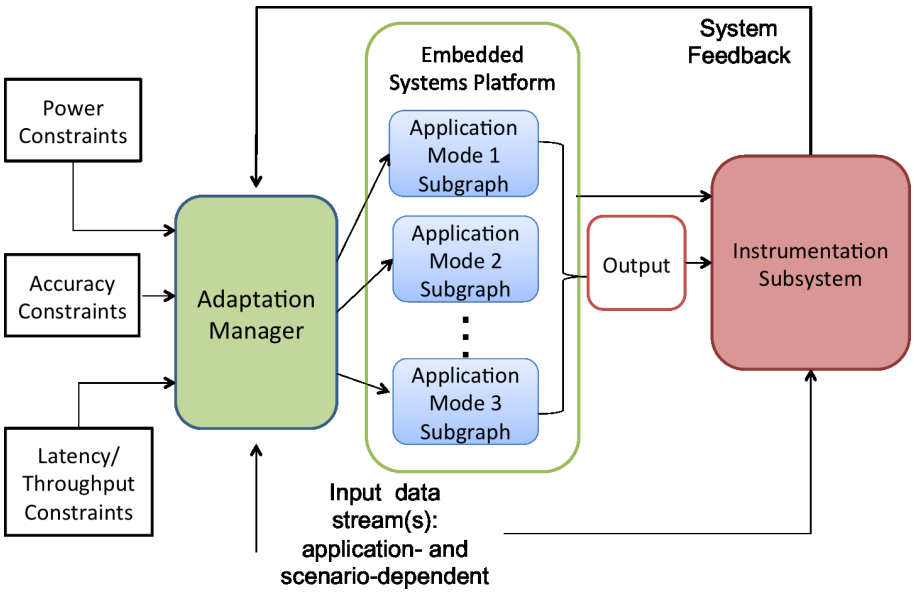

Figure 1.1: Data-driven, multi-mode embedded system design flow.

tation parameters based on measured data characteristics [3]. In this work, we combine the methodology of dataflow-based DSP system design with the DDDAS paradigm to address the novel constraints and challenges of real-time, multi-mode ASM processing on embedded platforms. Our proposed new design framework provides a structured approach for design, implementation and optimization of ASM systems under stringent platform constraints and dynamically-changing application requirements and data characteristics.

This report is written and organized based on the work published in [4]. To address the design and implementation of multi-mode ASM systems, we apply in this report our recently developed dataflow modeling technique called *Hierarchical Core Functional Dataflow* (*HCFDF*) [5]. In particular, we present a novel application of HCFDF to efficiently model and manage multi-mode application scenarios. In

this modeling approach, dynamic adaptation is represented through hierarchical inclusion of a special kind of actor (dataflow-based software component) called a *decision actor*. Such hierarchical use of decision actors is employed to switch among application subsystems based on data-driven demands involving performance-energy tradeoffs.

We also apply the HCFDF model to develop new methods for performance-energy-aware, dynamic scheduling of application subsystems. These scheduling techniques are geared towards efficient, context-aware adaptation of embedded DSP systems in multi-mode design scenarios. We integrate our new scheduling techniques with DDDAS concepts to introduce a unique model-based design environment for data-driven resource, constrained DSP applications. This design environment is prototyped and demonstrated by building on the *Lightweight Dataflow for Dynamic Data-Driven Application Systems Environment* (LiD4E), which is a tool for experimentation with and optimization of dataflow-based design methods for ASM systems [5].

Chapter 2

Background and Related Work

2.1 Core Functional Dataflow

Core functional dataflow (*CFDF*) is a dynamic dataflow model that provides highly expressive semantics for the design of applications with structured dynamic behavior [6]. In CFDF, an actor (dataflow graph functional component) is specified as a set of operational *modes*. In each mode, an actor consumes and produces fixed numbers of tokens on its input and output ports, respectively. These numbers of tokens consumed and produced are called the consumption and production *rates* of the associated input and output ports, respectively, and the associated modes. Consumption and production rates for CFDF actor modes can be arbitrary non-negative integers.

During execution, a CFDF actor operates in a unique *current mode* of the actor, which can be maintained as part of the actor state. Each actor has an associated *enable* function, which can be called by a run-time scheduler. The enable function returns a Boolean value indicating whether or not there is sufficient data available on the actor input ports to fire the actor in its current mode. The *invoke* function of an actor consumes data for execution based on the associated current mode. When an actor is invoked, it executes its current mode, produces and consumes data, and updates its current mode (i.e., sets the mode to be used in its next firing).

The enable function need not always be called before invoking an actor — in particular, it need not be called if static analysis of the graph can determine that the required data for the given actor mode will be available at the desired point of invocation. On the other hand, dynamic or quasi-static scheduling techniques may make use of the enable function to help ensure data availability in the absence of static guarantees [6].

2.2 Topological Patterns

For large-scale models of signal processing applications, the underlying dataflow graph representations often consist of smaller substructures that repeat multiple times. A method for scalable representation of dataflow graphs using topological patterns was introduced in [7]. Topological patterns, such as the *ring*, *butterfly*, and *chain* patterns, are pervasive in signal processing applications, including multi-dimensional signal processing systems, where processing of large scale dataflow structures is common. Topological patterns enable concise representation and direct analysis of sub-structures in the context of high level DSP specification languages and design tools. Modeling based on topological patterns also provides a scalable approach to specifying regular functional structures that is formally integrated with the framework of dataflow. This integration allows not only for specification of functional patterns, but also for their analysis and optimization as part of the larger framework of dataflow. For more details on modeling and design based on topological patterns, we refer the reader to [7].

2.3 Related Work

As mentioned in Chapter 1, the work presented in this report is rooted in core concepts of the DDDAS paradigm [3], and of dataflow-based design for DSP systems (e.g., see [8, 1]). In DSP-oriented dataflow modeling, applications are represented in terms of *dataflow graphs*, where graph vertices (*actors*) represent signal processing tasks of arbitrary complexity, and edges represent logical FIFO communication channels between pairs of actors. In this work, we apply dataflow as a programming model with semantics that are carefully matched to the targeted DSP application domain — i.e., dynamic, data-driven signal processing systems [1, 5], and more specifically, adaptive stream mining systems. This modeling approach differs from uses of dataflow as a compiler intermediate representation (e.g., see [9]), and as a form of computer architecture [10].

The work presented in this report builds upon our previous work on adaptive stream mining systems for multimedia applications [5]. The work presented in this report goes beyond our previous work by investigating design and implementation problems for multi-mode applications, and by developing new scheduling techniques for mapping applications onto embedded platforms while monitoring and managing dynamically-changing data characteristics and operational constraints.

Various studies on embedded stream mining have focused on performance optimizations for specific applications (e.g., see [11, 12, 13]). Similarly, the works of [13, 12] provide generalized scheduling and design strategies respectively, but focus on statically configured systems, without emphasis on handling time-varying data

characteristics. Work presented in this report is distinguished from these prior efforts in our focus on multi-mode application systems, and the integrated application of dataflow and DDDAS principles to such a multi-mode context.

Another relevant direction of prior work has involved the incorporation of data-driven adaptability to individual signal processing functional components (dataflow actors and their underlying algorithm parameters). For example, the works presented in [14, 15, 16] have studied such capabilities for speech processing applications. Here, adaptability is achieved by dynamically updating the key signal flow graph components, such as Hidden Markov Models (HMMs), linear predictive coding (LPC) blocks, and Mel-Frequency cepstral coefficients (MFCC) within a given speech recognition application [15, 14]. The methods can provide useful building blocks (parameterized actor and subsystem designs) for the directions that we pursued in [4]. However, the approach that we pursued in [4] is more flexible in terms of data-driven operation since we consider adaptation of application models globally (at the dataflow graph and scheduling level) as well as locally (at the level of individual actors or subsystems).

Chapter 3

Modeling Framework and Dynamic Multi-Mode Scheduling

3.1 Adaptive Classification

In this chapter, we demonstrate the utility of the LiD4E environment and the underlying HCFDF model of computation with a case study centering on an ASM system for face detection.

Adaptive classification systems are in general comprised of multiple classifiers with different parameters or performance goals (e.g., see [17]). In this context, we define "adaptive" as the ability of a system to change its operational parameters based on feedback or external input to maximize the effectiveness of the classification system, and select strategic combinations of classifiers dynamically through systematic processes to account for input data, operational constraints, and classifier characteristics. We define a "non-adaptive system" as one that operates using only one (static) set of parameters.

Support vector machines (SVMs) are supervised learning models that can be used for classification purposes (e.g., see [18]). A trained SVM classification model takes input data and calculates a value, which can then be thresholded to determine the class of the data. Conceptually, an SVM model is a representation of the training examples as hyperplanes with different classes separated by the widest gap allowed in the mapped space. The examples that are used to construct this

"maximum margin" are known as support vectors (SVs). Nonlinear classification using SVMs can be effective for the task of face detection [19]. One of the most popular kernels in this context is the Gaussian radial basis function, defined by $k(x_i, x_{SV}) = exp(-\gamma \|x_i - x_{SV}\|^2)$, where x represents the data point and γ is a parameter that can be configured. By using the kernel trick (i.e., the method for mapping observations to an inner product space), an SVM model can be trained efficiently on the data. Cross-validation can be used to determine optimal parameter values for each SVM based on the needs of the application.

3.2 Modeling Methodology

In this section, we discuss the modeling methods applied in our new design environment for data-driven DSP systems, and we demonstrate how they can be applied to the design of multi-mode application systems. These modeling methods are supported by the the LiD4E design tool, which is introduced in Chapter 1, and provides a foundation for our prototyping of and experimentation with the methods described in this report. While the underlying modeling foundation (HCFDF semantics) reviewed in this section has been developed in our previous work [5], our application of HCFDF semantics to multi-mode applications is a novel aspect described in this report.

A key feature of LiD4E is the provision for signal processing pipelines (i.e., chains of signal processing modules, such as classifiers, digital filters and transform operators) that can be data dependent and dynamically changing. LiD4E employs

hierarchical core functional dataflow (HCFDF) semantics as the specific form of dynamic dataflow [5]. Through its emphasis on supporting structured, application-level dynamic dataflow modeling, HCFDF provides a formal, model-based framework through which applications in DSP and related domains can be designed and analyzed precisely in terms of integrated principles of DDDAS and dataflow.

In HCFDF graphs, actors are specified in terms of sets of processing modes, where each mode has static (*dataflow rates*) — i.e., each mode produces and consumes a fixed number of data values (tokens) on each actor port. However, different modes of the same actor can have different dataflow rates, and the actor mode can change from one actor execution (*firing*) to the next, there by allowing for dynamic dataflow behavior (dynamic rates). Additionally, HCFDF allows dataflow graphs to be hierarchically embedded (nested) within actors of higher level HCFDF graphs, thereby allowing complex systems to be constructed and analyzed in a scalable manner. The design rules prescribed for hierarchical composition in HCFDF graphs ensure that actors at each level in a design hierarchy conform to the semantics of HCFDF or some restricted subset of HCFDF semantics, such as cyclo-static dataflow (CSDF) or synchronous dataflow (SDF) [20, 21]. For further details on HCFDF semantics, we refer the reader to [5].

As demonstrated in [5], HCFDF modeling enables run-time adaptation of signal processing topologies, including dataflow graphs that are constructed using arbitrary combinations of classifiers, filters, and transform units. Through the inclusion of a special HCFDF design component called an *adaptive classification module* (*ACM*), the designer can invoke multiple operating modes at run-time, and selection

of such operating modes can be driven based on system feedback — e.g., based on instrumentation that monitors data characteristics, and guides selection based on desired trade-offs among performance, accuracy, and energy consumption.

To apply such a hierarchical, DDDAS-based dataflow design methodology to the multi-mode application scenarios described in this report, we represent a system design as a set of mutually exclusive application modes $S_M = \{\mu_1, \mu_2, \ldots, \mu_N\}$, where each μ_i represents a set of application subsystems that are active during the corresponding mode together with the configurations (actor-, application- and schedule-level parameters) that are to be applied to the subsystems whenever μ_i executes. This is illustrated in Figure 3.1. Although execution across the μ_is is carried out sequentially, based on an ordering that can be determined dynamically, execution within each μ_i can consist of concurrent executions of an arbitrary number of HCFDF-based subsystems (dataflow subgraphs), and parallelism can be exploited within and across these concurrently executing subsystems.

Additionally, in our proposed design environment, the μ_is can share HCFDF subgraphs among them to promote code reuse, and reduce program memory requirements. For example, if a common speech processing subsystem is invoked in multiple application modes, it can be referenced from each of those modes, while having separate parameter settings, if desired, across the different modes that employ it. This leads, for example, to a design representation of information fusion alternatives as parameterized subsets of dataflow subgraphs, where each subgraph can be specialized to a particular type of information source (e.g., image, video, network event streams, speech, or high fidelity audio).

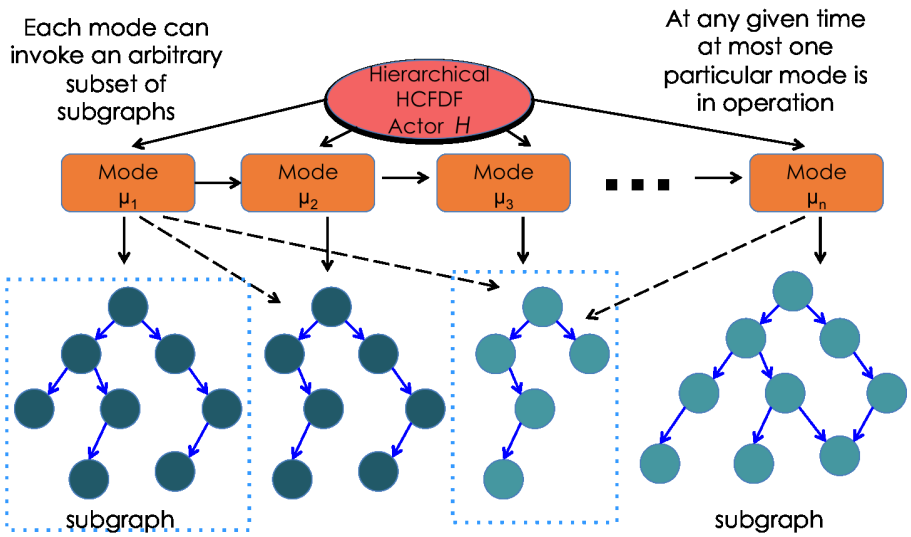

Figure 3.1: Modeling multi-mode DDDAS designs using HCFDF graphs.

3.3 Dynamic, Multi-Mode Scheduling

To integrate system-level, dynamic, data-driven operation into the targeted class of signal processing applications, we develop in this section an adaptive scheduling strategy for dynamic configuration and scheduling of multi-mode HCFDF graphs. The scheduling approach developed here is capable of dynamically adapting the selected application mode (e.g., high performance, high accuracy processing versus low energy, approximate processing) based on the overall health status of the target platform (e.g., available battery capacity), as well as on the data processing scenario (e.g., high-performance, alarm-driven scenarios versus low energy, standby scenarios).

The general scheduling approach, which we call the *DHMM* (*DDDAS-HCFDF Multi-Mode*) scheduler, involves a set of measurements $m_1, m_2, \ldots, m_k$ — from the target platform, operating environment or system output — that are to be taken

at discrete times during execution. Here, each measurement m_i corresponds to a distinct metric (e.g., instantaneous power consumption, remaining battery capacity, selected frequency content values for some kind of sensor data, or processing resource utilization as a percentage of available platform resources, to name a few possibilities). A natural way to schedule these measurements is just after each iteration of the executing application mode, since *dataflow graph iteration* is a commonly used concept of time window in the analysis of signal processing oriented dataflow programs (e.g., see [1]). Here, an application iteration can be defined to mean the processing period for a set of data frames (e.g., with one frame associated with each monitored sensor) for the current application mode, or can be parameterized to cover some number F of frame sets, where F can be adjusted dynamically to control to trade-offs among measurement overhead, adaptation frequency, and reactivity (the speed with which system reconfiguration can track changes in the measured data).

The sequence of measurement vectors, $\{(m_1(i), m_2(i), \ldots, m_k(i)) \mid i = 1, 2, \ldots\}$, obtained by this application-iteration-level instrumentation process drives a state machine S_{DHMM}, where the states are in one-to-one correspondence with the application modes, and each state σ has an associated function (i.e, a computational function, not just a mathematical function) f_σ. The purpose of each function f_σ is to compute parameter values for the mode associated with the state σ based on the newly observed instrumentation data (measurement vector), and any state variables that are maintained for σ. The state machine S_{DHMM} thus plays a central role in relating the *instrumentation subsystem*, which generates the measurement vectors, to the available application modes and their underlying dataflow subgraphs.

The design of the instrumentation subsystem — including selection of the metrics $\{m_i\}$ — along with the design of the state machine S_{DHMM} are important aspects of our overall adaptive scheduling methodology. The instrumentation subsystem and S_{DHMM} together with the HCFDF-based application- and mode-level dataflow subgraphs that they control lead to a precise, formally rooted, and platform-independent design framework for integrating DDDAS, dataflow, and multi-mode signal processing principles. In Chapter 4 we concretely demonstrate the DHMM scheduling methodology on a multi-modal, DDDAS-driven, design and implementation case study involving image processing.

We would like to emphasize that the objective of the DHMM scheduling methodology is not to introduce a new specialized scheduling algorithm for mapping dataflow graphs but rather to provide a systematic framework with which different schedules or scheduling algorithms can be integrated to provide DDDAS-driven, multi-mode integration for collections of signal processing subsystems (dataflow subgraphs). In particular, the "mode-level schedules" that are are used to execute specific application modes under specific mode parameter settings are *not* part of the DHMM framework specification. Such schedules can be derived by hand, statically by a software synthesis tool, at run-time or using a combination of synthesis-time and run-time techniques. Such separation of concerns between scheduling and system specification is a fundamental objective for dataflow-based signal processing environments (e.g., see [1]), and for model-based design tools in general.

Chapter 4

Case Study: Face Detection

4.1 Application Design and Experimental Setup

In this case study, we experiment with three SVM classifiers designed with different performance goals: high accuracy, low runtime, and low false positive rates. This experimentation is carried out through HCFDF-based modeling of multi-modal SVM classification using the three classifiers in conjunction with the framework of Figure 4.1, and dynamic selection of the classifier to use based on situational goals. We design, implement, and experiment with this ASM system using the LiD4E environment.

In this section, we demonstrate our proposed multi-mode, DDDAS-driven design approach, and our associated DHMM scheduling framework with a multi-mode application case study involving face detection. The metric vector that we consider in the instrumentation subsystem consists of a single component m_1, which corresponds to battery capacity, and the state machine S_{DHMM} is designed to gradually trade-off processing accuracy for energy efficiency as battery capacity drops from full to empty. Thus, we demonstrate how the targeted embedded system adapts in response to periodically measured data on system health, along with an underlying model of the design space across alternative classifier configurations.

Our experiments were performed through simulation on an Intel Core i7-2600K

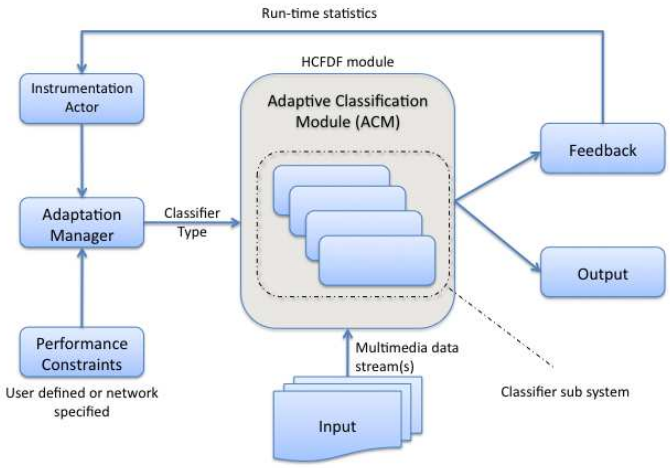

Figure 4.1: Lightweight Dataflow for DDDAS Envrionment

CPU (3.40GHz, 15GB RAM) running the Ubuntu 12.04 LTS operating system. The simulation — including HCFDF-based functionality for the DHMM scheduler, multi-mode application subsystems, and instrumentation subsystem — was implemented using the LiD4E environment [5]. In particular, the C-based application programming interfaces (APIs) of LiD4E were employed; thus, our experimental system implementation can be viewed as a C language realization that employs LiD4E APIs to achieve the desired forms of high level dataflow semantics.

The experiments reported on in this section can be viewed as providing initial demonstration and validation of the multi-mode, DDDAS design methodology presented in this report. More complex experiments — e.g., involving multi-dimensional instrumentation spaces (metric vectors with multiple components) and implementations on embedded platforms — are a useful direction for future work.

The face detection application that we experiment with in this report is based

on an application introduced in [5], with modifications incorporated to integrate the DHMM scheduling framework with the metric m_1 described above for battery capacity, and a state machine S_{DHMM} that is designed to provide decreasing levels of processing accuracy and energy consumption as the battery level decreases. These alternative accuracy/energy trade-offs are captured discretely through three separate states (application modes) in S_{DHMM}. The three modes correspond to three distinct classifier configurations, which can be viewed, respectively, as configurations that provide maximum energy efficiency; a trade-off among accuracy, energy efficiency, and false positive rates; and a minimum false positive rate. We refer to these modes as M_1, M_2, and M_3. Here, energy efficiency is measured in terms of the amount of energy consumed per classification operation (mode invocation). Thus, M_1 has the lowest energy consumption, M_3 has the highest, and M_2 has an intermediate level of energy consumption.

The accuracy and false positive rates for a set of executed classification operations are defined, following standard convention, as follows. Suppose that C classification tasks are performed, and among these, c_1, c_2, c_3, c_4 tasks represent the true positives, true negatives, false positives, and false negatives ($c_1 + c_2 + c_3 + c_4 = C$), respectively. Then we define the associated classification *accuracy* as $(c_1 + c_2)/C$, and the *false positive rate* (*FP rate*) as c_3/C. In many kinds of operational scenarios — e.g., where FPs are much more costly compared to false negatives — the FP rate is viewed as being more important than maximizing accuracy (at least up to some allowable degradation in accuracy).

The scheduler state machine S_{DHMM} is parameterized with a two-element vec-

tor, $\nu \in V$, called the *threshold vector*. Here, V, the set of permissible values for ν, is defined by $V = \{(x, y) \mid 0 \leq y \leq x \leq 1\}$. Given an initial battery capacity J, transitions between modes are carried out in S_{DHMM} by starting initially in M_3, transitioning to M_2 once the battery capacity falls below $x \times J$, and then transitioning to M_1 (the most energy efficient mode) once the battery capacity falls to $y \times J$. Certain boundary conditions in V lead naturally to special cases in the trajectory of modes. For example, if $x = 1$, then we transition immediately to M_2, and if $x = y$, then M_2 is effectively skipped.

In our experiments, we use $F = 1$ as the iteration length (see Section 3.3), meaning that the DHMM scheduler makes its next assessment about whether to switch modes after each new image is processed. The SVM classifier parameters for all three application modes were developed using MATLAB, trained using the MIT CBCL face database [22], and then ported to C in the LiD4E environment.

4.2 Experimental Results

In our simulation setup, we estimate the energy consumption of a classification task as being proportional to the latency, and we assume that the target platform consumes negligible energy consumption during idle periods through use of power-saving sleep capabilities. More specifically, we assume a constant average power consumption ρ across all modes so that the battery energy drained for a given mode invocation is estimated as $\rho \times \tau(\mu)$, where $\tau(\mu)$ is the average latency (processing time), as measured for mode μ. This model is used to simulate draining of the

battery from full capacity to empty capacity. This simulated draining process in turn creates a stream of battery capacity data, which is used to drive the DHMM adaptation process implemented by S_{DHMM}. This is a relatively simple model of energy consumption; application of more sophisticated energy models is an interesting direction for further work.

Table 4.1 shows experimental results for several different threshold vectors. The number of processed images (third column) gives a measure of the overall energy efficiency across the lifetime of the system (i.e., until the battery is fully drained). The three threshold vectors at the bottom (labeled SDF1, SDF2, and SDF3) each correspond to execution of a single mode for the entire input stream (no state transitions). Such implementations represent statically-structured implementations that do not employ multi-mode/DDDAS capabilities, and can be implemented as synchronous dataflow (SDF) graphs, without use of more dynamic features, including HCFDF modeling or the proposed DHMM scheduler.

Intuitively, the DHMM-based system provides a way to achieve configurable, graceful degradation of classification quality (accuracy and FP rate) as the battery expires. This can be important, for example, if a mission lasts significantly longer than expected. The results in Table 4.1 help to quantify this kind of graceful degradation, and also demonstrate another important advantage of the DHMM-based approach: the approach allows for finer-grained control over the overall design evaluation space (i.e., in this case, the space involving energy efficiency, accuracy, and FP rate). By varying the threshold vector, the designer or a run-time system can steer the overall system performance (across the entire mission) towards a specific

System	Threshold Vector	Number of Processed Images	Accuracy	False Positive Rate
DHMM	$\nu 1$(0.9,0.2)	1338	99.23%	17.68%
	$\nu 2$(0.7,0.4)	1545	98.53%	21.93%
	$\nu 3$(0.7,0.05)	901	97.83%	12.92%
	$\nu 4$(0.6,0.1)	915	97.26%	14.89%
	$\nu 5$(0.5,0.25)	1111	97.13%	19.73%
	$\nu 6$(0.1,0.05)	424	87.95%	10.18%
SDF1	(0,0)	2890	99.56%	26.19%
SDF2	(1.0,0)	1052	99.71%	11.64%
SDF3	(1.0,1.0)	256	78.31%	0%

Table 4.1: Experimental results

Pareto-optimal point in the space that represents the best trade-off for the application. Thus, rather than being confined by just the trade-offs provided by the individual classifiers (i.e., the last three rows in Table 4.1 for this case study), the designer or run-time system has a large amount of control in steering the overall performance into other regions of the underlying design evaluation space. These capabilities — configurable and graceful degradation and the production of new, Pareto-optimal operating alternatives — represent significant advantages derived by applying the multi-mode, DDDAS techniques discussed in this report.

Chapter 5

Conclusion

In this report, we have discussed an approach to design and implementation of multi-mode, data driven signal processing systems. We have developed methods for modeling and designing such systems using integrated principles of dynamic data driven application systems (DDDAS) and high-level, dynamic dataflow models of computation. We have introduced a scheduling framework, called the DHMM (DDDAS-HCFDF Multi-Mode) scheduler, for instrumentation-driven, adaptive scheduling in multi-mode signal processing systems. Through a case study of an energy-constrained, multi-mode face detection system, we have demonstrated and quantified significant advantages of our proposed new methods. Useful directions for future work include (1) extensions to multiple sensing modalities, such as integrated image and speech processing, and (2) experimentation with instrumentation subsystems that produce multidimensional outputs (e.g., channel quality in addition to power consumption).

Chapter 6

Acknowledgement

This research was sponsored in part by the Austrian Marshall Plan Foundation, and the DDDAS program under US Air Force Office of Scientific Research (AFOSR).

Bibliography

[1] S. S. Bhattacharyya, E. Deprettere, R. Leupers, and J. Takala, Eds., *Handbook of Signal Processing Systems*, Springer, second edition, 2013, ISBN: 978-1-4614-6858-5 (Print); 978-1-4614-6859-2 (Online).

[2] R. Ducasse and M. van der Schaar, "Finding it now: Construction and configuration of networked classifiers in real-time stream mining systems," in *Handbook of Signal Processing Systems*, S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takala, Eds., pp. 97–134. Springer, second edition, 2013.

[3] F. Darema, "Grid computing and beyond: The context of dynamic data driven applications systems," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 692–697, 2005.

[4] K. Sudusinghe, I. Cho, M. van der Schaar, and S. S. Bhattacharyya, "Model based design environment for data-driven embedded signal processing systems," in *Proceedings of the International Conference on Computational Science*, Cairns, Australia, June 2014, pp. 1193–1202.

[5] K. Sudusinghe, S. Won, M. van der Schaar, and S. S. Bhattacharyya, "A novel framework for design and implementation of adaptive stream mining systems," in *Proceedings of the IEEE International Conference on Multimedia and Expo*, San Jose, California, July 2013, 6 pages in online proceedings.

[6] W. Plishker, N. Sane, M. Kiemb, K. Anand, and S. S. Bhattacharyya, "Functional DIF for rapid prototyping," in *Proceedings of the International Symposium on Rapid System Prototyping*, Monterey, California, June 2008, pp. 17–23.

[7] N. Sane, H. Kee, G. Seetharaman, and S. S. Bhattacharyya, "Scalable representation of dataflow graph structures using topological patterns," in *Proceedings of the IEEE Workshop on Signal Processing Systems*, San Francisco Bay Area, USA, October 2010, pp. 13–18.

[8] E. A. Lee and T. M. Parks, "Dataflow process networks," *Proceedings of the IEEE*, pp. 773–799, May 1995.

[9] W. Najjar, B. Draper, W. Bohm, and R. Beveridge, "The cameron project: High-level programming of image processing applications on reconfigurable computing machines," in *Proceedings of the PACT Workshop on Reconfigurable Computing*, 1998.

[10] J. B. Dennis, "Dataflow supercomputers," *IEEE Computer Magazine*, vol. 13, no. 11, November 1980.

[11] U. Ramacher, "Software-defined radio prospects for multistandard mobile phones," *IEEE Computer Magazine*, vol. 40, no. 10, pp. 62–69, 2007.

[12] J. Pisharath, N. Jiang, and A. Choudhary, "Evaluation of application-aware heterogeneous embedded systems for performance and energy consumption," in *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, 2003, pp. 124–132.

[13] F. König, D. Boers, F. Slomka, U. Margull, M. Niemetz, and G. Wirrer, "Application specific performance indicators for quantitative evaluation of the timing behavior for embedded real-time systems," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, 2009, pp. 519–523.

[14] G. Chollet, K. McTait, and D. Petrovska-Delacrétaz, "Data driven approaches to speech and language processing," in *Nonlinear Speech Modeling and Applications*, pp. 164–198. Springer, 2005.

[15] G. Aradilla, J. Vepa, and H. Bourlard, "Improving speech recognition using a data-driven approach," Tech. Rep. IDIAP-RR 05-66, IDIAP Research Institute, April 2005.

[16] H. Ney, D. Mergel, A. Noll, and A. Paeseler and, "Data driven search organization for continuous speech recognition," *IEEE Transactions on Signal Processing*, vol. 40, no. 2, pp. 272–281, 1992.

[17] R. Ducasse, D. Turaga, and M. van der Schaar, "Adaptive topologic optimization for large-scale stream mining," *IEEE Journal on Selected Topics in Signal Processing*, vol. 4, no. 3, pp. 620–636, June 2010.

[18] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Knowledge Discovery and Data Mining*, vol. 2, no. 2, 1998.

[19] B. Heisele, P. Ho, J. Wu, and T. Poggio, "Face recognition: component-based versus global approaches," *Journal of Computer Vision and Image Understanding*, vol. 91, no. 1–2, pp. 6–21, 2003.

[20] G. Bilsen, M. Engels, R. Lauwereins, and J. A. Peperstraete, "Cyclo-static dataflow," *IEEE Transactions on Signal Processing*, vol. 44, no. 2, pp. 397–408, February 1996.

[21] E. A. Lee and D. G. Messerschmitt, "Synchronous dataflow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235–1245, September 1987.

[22] "CBCL face database #1," `http://cbcl.mit.edu/software-datasets/FaceData2.html`, 2010.