Instrumentation-Driven Model Detection and Actor Partitioning for Dataflow Graphs

Ilya Chukhman, Department of Electrical and Computer Engineering, Institute for Advanced Computer Studies, University of Maryland, College Park, MD, USA

Shuoxin Lin, Department of Electrical and Computer Engineering, Institute for Advanced Computer Studies, University of Maryland, College Park, MD, USA

William Plishker, Department of Electrical and Computer Engineering, Institute for Advanced Computer Studies, University of Maryland, College Park, MD, USA

Chung-Ching Shen, Department of Electrical and Computer Engineering, Institute for Advanced Computer Studies, University of Maryland, College Park, MD, USA

Shuvra S. Bhattacharyya, Department of Electrical and Computer Engineering, Institute for Advanced Computer Studies, University of Maryland, College Park, MD, USA

ABSTRACT

Dataflow modeling offers a myriad of tools to improve optimization and analysis of signal processing applications, and is often used by designers to help design, implement, and maintain systems on chip for signal processing. However, maintaining and upgrading legacy systems that were not originally designed using dataflow methods can be challenging. Designers often convert legacy code to dataflow graphs by hand, a process that can be difficult and time consuming. In this paper, the authors developed a method to facilitate this conversion process by automatically detecting the dataflow models of the core functions from bodies of legacy code. They focus first on detecting static dataflow models, such as homogeneous and synchronous dataflow, and then present an extension that can also detect dynamic dataflow models. Building on the authors' algorithms for dataflow model detection, they present an iterative actor partitioning process that can be used to partition complex actors into simpler sub-functions that are more prone to analysis techniques.

Keywords: *Classification, Dataflow Graphs, Instrumentation, Models of Computation, Signal Processing Systems*

DOI: 10.4018/jertcs.2013010101

INTRODUCTION

Modern digital signal processing (DSP) systems run sophisticated algorithms on high-performance platforms based on FPGAs, programmable digital signal processors (PDSPs), and multiprocessor system-on-chip (MPSoC) devices. As a result, designing these systems is a complex process prone to inefficiencies and mistakes. Design tools, including dataflow modeling, are often used to help with the design process. Modeling DSP applications through coarse-grain dataflow graphs is widespread in the DSP design community, and a variety of dataflow models have been developed for dataflow-based design (DBD). DBD allows a designer to decompose a complex system into simpler sub-functions (*actors*) that are connected to form a graph. A variety of dataflow modeling tools can then be used to verify correctness of the graph and optimize the entire system, for instance see (Lee & Messerschmitt, 1987; Buck, 1993; Siyoum, Geilen, Moreira, Nas, & Corporaal, 2011; Plishker, Sane, Kiemb, Anand, & Bhattacharyya, 2008).

When employing DBD techniques, it is useful for a designer to find a match between his actors and one of the well-studied models, such as homogeneous synchronous dataflow (HSDF), synchronous dataflow (SDF) (Lee & Messerschmitt, 1987), cyclo-static dataflow(Bilsen, Engels, Lauwereins, & Peperstraete, 1996), or Boolean dataflow (BDF) (Buck, 1993). When such a match is found, one can systematically exploit specialized characteristics of actors that conform to the models, and take advantage of more effective, model-specific methods for analysis and optimization. For example, if a dataflow model match cannot be found, a less efficient, generic scheduler and more conservative memory allocation may need to be employed.

Economic factors necessitate reuse of existing designs with periodic upgrades to keep up with technological advances while saving on the non-recurring engineering costs associated with new designs. For example, the Large Hadron Collider (LHC) used for high energy physics experiments is planned to undergo a periodic series of large technology upgrades to allow for new experiments and the expansion of existing experiments(Gregerson, Schulte, & Compton, 2010). Having a dataflow representation of such a system can alleviate this upgrade process by facilitating correctness verification, and in some cases enabling the use of automatically generated implementations for the new hardware(Miyazaka & Lee, 1997; Oh & Ha, 2002). DSP systems that are not designed using DBD, including legacy systems, are more difficult to upgrade, since implementation details can lead to errors that are hard to detect. For this reason, deriving dataflow graphs for these systems is beneficial and is increasingly done even though converting existing DSP code to dataflow graphs can be difficult and time consuming (e.g., see (Bhattacharyya, Deprettere, Leupers, & Takala, 2010)).

To implement and experiment with our proposed model detection methodology, we have employed the DSPCAD Integrative Command Line Environment (DICE) (Bhattacharyya, Plishker, Shen, Sane, & Zaki, 2011), which is a framework for facilitating efficient management of design and software projects. DICE defines platform- and language-agnostic conventions for describing and organizing tests, and uses shell scripts and programs written in high-level languages to run and analyze these tests.

To create a generic method for instrumenting dataflow graphs, we used a DBD framework called the Lightweight Dataflow Environment (LIDE) (Shen et al., 2011), which is supported by DICE. This framework supports dynamic dataflow applications with a semantic model called core functional dataflow (CFDF) (Plishker, Sane, Kiemb, Anand, & Bhattacharyya, 2008). From its foundation in CFDF semantics, LIDE enables dynamic behavior through structured application descriptions, making it an effective platform to instrument dataflow graphs, and prototype techniques for automated dataflow model detection.

As with unit testing, we rely on significant *coverage* of the behaviors of an actor instead of requiring a formal solution to analyze the code of the actor itself. While this requires