High-Level Architecture Modeling and Exploration for Streaming Applications

Mirza, Usman Mazhar; Gruian, Flavius

Published in:
Proceedings of the Forty Sixth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)

DOI:
10.1109/ACSSC.2012.6489226

2012

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.
Abstract—Multi and many core platforms are increasingly employed today to cope with the demands of modern multimedia, networking and other similar streaming applications. Selecting the best architecture for a set of such applications is complex, and without coding for each specific system, the choice is made based on designer experience. In this paper we use CAL, a dataflow language for specifying streaming applications, to model both the software and generic hardware platforms. The hardware model is combined with the mapped application in a high-level unified model, which is simulated to obtain performance estimates, intended to give feedback during design space exploration.

I. INTRODUCTION

Streaming applications (e.g. multimedia, networking and radar signal processing) are today ubiquitous and become increasingly demanding. In order to meet the computational and parallelism demands of today’s streaming applications, multi and many core platforms are now replacing uniprocessor systems. A survey of current such platforms reveals there are considerable differences between these in terms of processing elements and interconnect (point-to-point, bus-based, packet-based network-on-chip (NoC)). Therefore, each requires a specific programming solution, which makes it difficult to evaluate the performance of an application running on a platform before an actual implementation. Without a working implementation, selecting the best architecture and tuning its parameters (number and type of processing elements, interconnect bandwidth, arbitration, routing algorithm, switching technique, scheduling, etc.) for a specific set of applications is a matter of experience or educated guesses.

Dataflow is a natural way of modeling streaming applications, with its synchronous, cyclo-static and dynamic flavors [9], [2]. CAL is a dataflow language [5], based on communicating concurrent actors, successfully adopted as MPEG/ISO standard in the form of RVC-CAL. Furthermore, there is a strong tool support for CAL, including simulators, C and VHDL backends, as well as benchmark applications.

In our study we envision the system design process as an iterative activity, comprising architecture selection, mapping and scheduling, followed by evaluation leading back to architecture tuning (see Fig. 1). For any meaningful design space exploration, this evaluation must be fast and preferably automatic. Therefore, we are interested in fast, high-level simulation of streaming applications on a generic, tunable architecture model.

This paper focuses on applications specified in CAL, and in particular in highly dynamic applications, for which static analysis and scheduling are imprecise. In particular, we are interested in applications mapped to many-core platforms, where there exists the choice of selecting and changing the hardware architecture. Our work in this paper uses a compound model (specified in CAL) of the hardware and software that can be used for obtaining performance parameters, such as latency, throughput, and energy consumption. These performance parameters are intended as feedback during design space exploration.

The remainder of our paper is structured as follows. In the next section we discuss related work, while in section III we describe our view of the design flow in detail and present an overview of CAL. In section IV we describe architecture modeling with CAL. Section V presents the evaluation of our modeling technique. Finally, we conclude our paper in section VI and mention future work.

II. RELATED WORK

Performance evaluation of streaming applications running on many-core platforms and selecting the best architecture for a specific set of streaming applications, before its actual implementation, are complex problems and interesting for both industry and academia. Two existing methods for this purpose are analytical analysis and simulation. There are number of research papers published in both areas. We will mention few of them here.

Many analytical methods exist for evaluating many-core networked platforms where the difficult problem is the interconnect. An evaluation method to analyze the delay of wormhole routing based different heterogeneous NoC architectures with variable link capacities and variable number of virtual channels per link is presented in [1]. That work addresses the performance evaluation of NoC based interconnect structures only. In [3], a framework for cache-aware timing analysis of streaming applications is presented. This method seamlessly integrates program analysis techniques for micro-architectural modeling with known analytical methods for analyzing streaming applications. Performance analysis of streaming applications on multiprocessor systems-on-chip is also discussed in detail in [7] and [11].

Many simulators are also developed to model streaming applications running on many-core platforms and successfully...
used for performance estimation. A micro-architecture simulator for a stream processor is implemented in [10]. It evaluates the performance of streaming applications and analyzes the utility rate of hardware and memory consumption. In [8], a fast modular trace-based simulation framework is presented for performance estimation of streaming applications on complex MPSoC architectures. That simulator is developed in SystemC. Another work [6], describes the simulation of streaming applications on multicore systems using X-Sim (simulation component) of Auto-Pipe development system.

In our work we are using CAL dataflow language for modeling the architecture. As the streaming applications are also specified in CAL, it is easy to interface the architecture with the application. The motivation behind the modeling of architecture in CAL is that dataflow analysis techniques can be applied to our compound model (hardware and software) to estimate different design parameters. An alternative would be modeling the architecture in SystemC and translate the mapped actors into C using a concurrency model supported by each platform. Nevertheless, we considered this to have more drawbacks than advantages in our case. Using already available SystemC models for specific architectures also means using different programming interfaces, making the translation of actors to viable C code problematic. Additional problems would result from using mixed language simulation, for architectures that offer simulators written in other languages (VHDL/Verilog, Java) and supporting all of these would not be feasible.

III. OUR VIEW OF THE SYSTEM DESIGN FLOW

We consider the system design process, as shown in Fig. 1 as an iterative activity. In the design flow, after the selection of hardware platform (architecture) the streaming application is mapped and scheduled on the selected architecture. This step is followed by evaluation leading back to tuning of architecture and mapping.

Both the hardware platform and mapped application are modeled, and this compound model is then simulated to get performance estimates such as latency, throughput and energy consumption. The estimated performance parameters are used as feedback metrics for tuning the architecture as well as the mapping. The whole procedure of performance estimation is usually referred to as design space exploration. For fast evaluation we are interested in high-level simulation of streaming applications on a generic, tunable hardware platform model.

A. CAL Overview

CAL is a dataflow language [5] designed to describe streaming algorithms based on computational kernels (actors) and communication lines (channels) between actors. Actors and channels provide a powerful support for modeling and programming streaming applications on a high level of abstraction. An actor in CAL is a modular component and it encapsulates its own state. It has a number of input/output ports, internal state, set of executable actions and set of rules (action priorities, state transition definitions, guard conditions) that govern actions’ execution. The functionality of the actor is defined by the set of actions where each action is atomic and its execution is called a firing. The interaction between actors, sending and receiving tokens, is only possible through the channels (FIFO buffers) that connect output port(s) of one actor with input port(s) of another.

An action can only fire if it has enough tokens on all of its input ports, its guard conditions (if it has any) are satisfied and if it has the highest priority (in case of multiple actions that are eligible to fire at the same time). An action may perform any of the following steps during its firing:

- Consume tokens from its input ports
- Modify actors internal state
- Produce tokens at its output ports

CAL supports hierarchical networks of actors and provides a mechanism for encapsulation and modularization for building complex systems. An application in CAL usually consists of several networks of actors. A number of CAL actors are grouped together in a network using a network language (NL). The two steps for building a network of actors are:

- actors instantiations, and
- specifications of the connections between their ports.

A network of actors may be viewed as an actor and can itself have input and output ports, to support hierarchy, which may be connected to the ports of actors inside it.

We are using CAL dataflow language in our study as a modeling language because there is a strong tool support available for CAL. Furthermore, timing annotations for actions (actor firings) are supported in CAL, which allows for high-level performance evaluation.

IV. ARCHITECTURE MODELING WITH CAL

Our method uses CAL as modeling language for modeling both generic hardware platform and software applications. As
illustrated in Fig. 2 we see top-level composite (architecture + application) model as a hierarchical network of CAL actors. Each processing element and interconnect structure is a separate network of actors, that in turn may be viewed as an actor. These actors, are composed into a top-level network model.

The interconnect structure modeled so far is a very simple mesh network that uses Manhattan distance (units) between processing elements (PEs) as the communication delay. The assumptions made for modeling of the architecture are presented in the next section.

An example of processing element modeled as CAL actor is shown in Fig. 3. It is composed of mapped actors (application partition) and in/out interfaces. The in/out interfaces are again CAL actors that describe how the input/output ports of processing element (upper level actor) are connected with mapped actors on that processing element. It also enables their communication with the outer environment (i.e. interconnect structure and other PEs).

One important thing to notice here is once the mapping of actors to the processing elements is decided, the top-level model is generated automatically with the actors mapped on respective processing elements. We do not need to modify the original application actors but processing elements just encapsulates them. The composite CAL model, see Fig. 2, is then profiled providing timing (and power/energy) figures that allow to either tune the architecture or find a better mapping.

Our intention is to use parameterizable models, that in future may be tuned to emulate a number of different interconnect structures and processing elements. Using such models, a range of architectures can be generated, for which the number and types of processing elements and interconnect structures vary, resulting in a platform library.

V. EVALUATION

To evaluate our modeling technique, we used three benchmark applications Mandelbrot, GameOfLife and FIR (see Table I) implemented in CAL. These applications are taken from OpenDF repository [1]. In our experiments we showed that combined (hardware + software) models, specified in CAL, can be generated automatically once the mapping of application on different architectures is provided. These automatically generated models can be simulated to get performance parameters such as latency, throughput and power consumption. At this point we just model the communication delay, in cycles, of interconnect structures and therefore currently we are only considering the average communication latency as performance parameter for the interconnect. We show that different mappings of actors to processing elements result in different average communication latency. Before going into the details of simulation results using our automatically generated models, we will mention the number of assumptions that we made during design of our preliminary unified model.

As hardware platform, we assumed mesh (1x4, 2x2, 3x3) platforms with homogeneous PEs. Tokens (communicating packets) are assumed to be of same size and the link bandwidth for all links is assumed to be equal to the size of packet. The communication delay between PEs in the interconnect structures is assumed as the Manhattan distance (units) between them. Scheduling inside PEs and network congestion are not modeled at this point.

Average communication latency (average packet latency) is the average delay of network to transfer payload from all source PEs to all destination PEs. We used the above mentioned streaming applications and mapped them on different architectures in our unified model. We then run the simulations for a specific number of packets and recorded the average latency.
TABLE I
STATISTICS OF DIFFERENT STREAMING APPLICATIONS MAPPED ON DIFFERENT ARCHITECTURES.

<table>
<thead>
<tr>
<th>Application</th>
<th>No. of Actors</th>
<th>Architecture Mapping</th>
<th>No. of Actors (System)</th>
<th>Average Packet Latency</th>
<th>No. of Simulated Packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mandelbrot</td>
<td>5</td>
<td>2x2 A</td>
<td>28</td>
<td>1.556</td>
<td>40000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2x2 B</td>
<td>25</td>
<td>1.333</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1x4 A</td>
<td>28</td>
<td>1.667</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1x4 B</td>
<td>25</td>
<td>1.333</td>
<td></td>
</tr>
<tr>
<td>GameOfLife</td>
<td>3</td>
<td>2x2 -</td>
<td>19</td>
<td>1.508</td>
<td>20000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1x4 -</td>
<td>19</td>
<td>2.016</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2x2 -</td>
<td>32</td>
<td>1.333</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3x3 A</td>
<td>49</td>
<td>2.030</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3x3 B</td>
<td>49</td>
<td>2.182</td>
<td></td>
</tr>
</tbody>
</table>

packet latency in cycles. The results of our simulations are presented in Fig. 4. These results are also presented in tabular form in Table I. The table also shows the number of simulated packets and the number of actors for each application before and after transformation to the combined CAL model. We can see that mapping any streaming application on different architectures results in different average communication latency. For example, in case of FIR, the average communication latency if we map FIR on 2x2 and 3x3 mesh platforms is 1.333 and 2 respectively. We can also notice that different mappings of the same streaming application can also result in different average communication latency. As in the case of Mandelbrot, two different mappings (A, B) of application on 2x2 mesh platform resulted in 1.556 and 1.333 average communication latency.

As a very next step we plan to work on detailed model of different interconnect structures (point-to-point, buses, NoC) with delays involved in routing, contention and arbitration. We also want to model the scheduling of mapped actors on a processing element and use system level performance parameters such as latency, throughput, energy consumption to provide feedback during design space exploration. Finally we will validate our approach by comparing the simulation results from our combined model with the figures obtained by compiling and running the same applications on real physical platforms (e.g. Ambric, STM P2012).

ACKNOWLEDGMENT
This work has been supported by the Swedish Foundation for Strategic Research (SSF).

REFERENCES