A Performance Modeling and Analysis Environment for Reconfigurable Computers

Jeffrey Walrath and Ranga Vemuri
email : ranga.vemuri@uc.edu

University of Cincinnati, ECECS Department, ML #0030,
Cincinnati, OH 45221, USA

Abstract. In many of the various layers of software supporting reconfigurable architectures such as compilers, operating systems, synthesis tools, and so forth, a primary objective is to deliver the performance, power, cost, and other advantages of reconfigurable architectures to a target application. Inherent to these tools are various estimation procedures for such performance metrics as throughput time, power, reliability, cost, and so on. Analysis of Reconfigurable Computers (ARC) is a comprehensive analysis and modeling tool we are developing that can be used to calculate these and other performance metrics.

1 Introduction

Currently, there is significant research and development in reconfigurable architectures from compilers to automated synthesis tools to various programming methodologies and abstractions [1, 2]. It is common to find in these tools estimation techniques for various performance metrics such as execution time, power consumption, throughput rate, reliability, etc.

Since this type of estimation is such a common thread, we are developing a comprehensive performance modeling and analysis environment specifically for the analysis of reconfigurable computers called ARC [1]. Figure 1 shows an application-level view of the ARC system (Analysis of Reconfigurable Computers). It will allow for, but not be limited to, the analysis of various performance metrics such as throughput rate, reliability, power consumption, reconfiguration costs, and so forth. And by the nature of its design, it will be possible with ARC to perform analysis of any computable performance metric that can be specified in our performance model specification.

ARC is a modular system designed for flexibility in doing performance modeling and analysis of various performance metrics on a number of different reconfigurable architectures including both hardware and software architectures. Inputs to the ARC system are an Architecture Specification and a Performance Model Specification. The architecture specification is a description of the target architecture. A performance model specification defines the performance metrics

[1] This work is sponsored in part by the DARPA/ITO Adaptive Computing Systems Program and monitored by the US Army under contract number DABT63-97-C-0030.

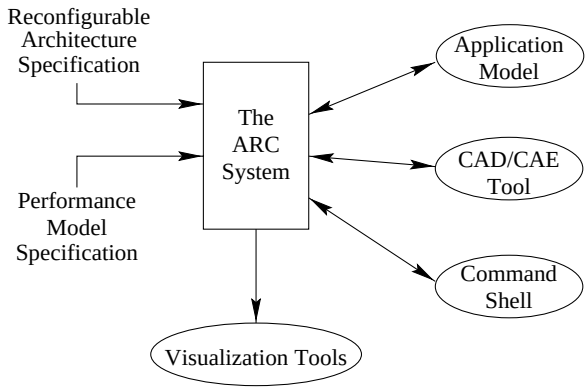

Fig. 1. View of the ARC system.

to be analyzed and procedures for calculating those metrics. Performance model specifications are given in a performance modeling language based on PDL [3, 4].

Since the architecture and performance model are given as separate inputs, the ARC system can be easily modified and configured to perform different performance analyses by simply changing the performance model specification. In addition, a performance model can be written so that it applies to a family of architectures instead of just a single architecture. For example, a performance model can be written to calculate the throughput time of designs configured on an FPGA style architecture. One performance model specification could be written so that it applies to an entire family of FPGAs.

With these two inputs, the ARC system internally creates a flexible executable performance model for the specified architecture. It is flexible because it can be used for any configuration of the target architecture. That is, ARC automatically updates and modifies the performance model each time a new configuration is given. Thus, the ARC system can be used to analyze various performance metrics over a number of different configurations. It maintains and stores data and various information contained in the model in a results database that can then be used by other visualization tools and environments such as Khoros or Gnuplot.

Figure 2 shows a system-level view of ARC and the various components of the system. In the following sections, we describe the details of the inputs to the ARC system along with the operation of the various components.

2 Performance Model Specification

A performance model specification defines both the performance attributes to be calculated along with the evaluation rules for calculating values for the various attributes. From a modeling perspective, a design is viewed as a collection of *modules*, *carriers*, and *ports*. Thus, all the components of an actual design can be categorized as one of these three types. Modules are typically such items as gates, logic blocks, switch boxes, transistors, etc. Carriers are typically wires and

net-lists. Ports are used for connecting modules and carriers to other modules and carriers. These component types are declared in the model specification.

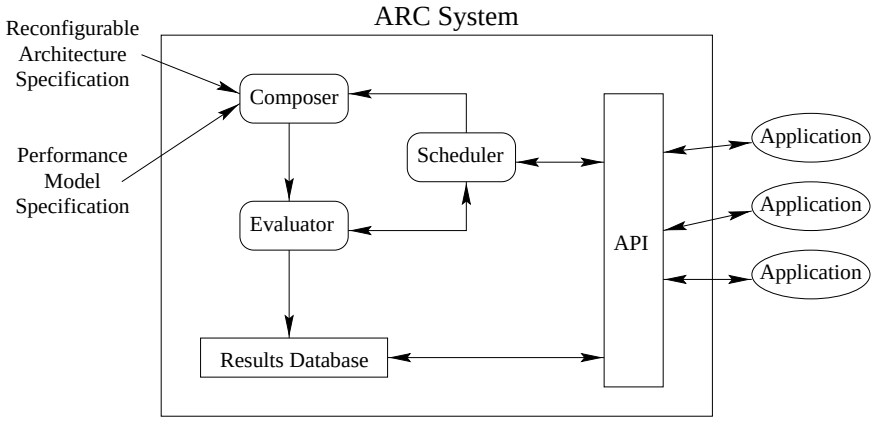

Fig. 2. Components of the ARC system.

Along with the component types are definitions for the various attributes and evaluation rules. Attributes are defined as a particular data type such as integer, real, etc. and attached to the different component types [5, 6]. Evaluations rules define how to calculate various performance metrics and aspects. These rules are composed of expressions, functions, and programming statements that may involve the various attributes of the different design components.

It is important to note here that a performance model specification is defined independent of a specific design. Instead, it is defined for a family of designs. That is, the same specification can be applied to any design that contains those design components declared in the model specification.

Design components, attributes, and evaluation rules are written in a language based upon the performance modeling language PDL [3, 4]. This modeling language is flexible enough to allow for the analysis of any computable performance metric that can be defined in this notation.

2.1 Attribute Types

Another important aspect of the model specification is attribute types. Besides being a particular data type, attributes can also be *static* or *dynamic*. Static attributes are those attributes in the model which can be computed from the static structure of the design being modeled. They can include such things as area, maximum delay throughput, supply voltage, and so forth. Dynamic attributes are those attributes that are calculated based on the dynamic behavior of the design. For example, they can include such things as dynamic power dissipation, size of queues and fifo, etc.

Unlike static attributes, dynamic attributes require streams of values. Some dynamic attributes are streams of values specified externally via the API while

other dynamic attributes depend upon the values of other dynamic attributes within the model. Evaluation rules for dynamic attributes can reference both static and dynamic attributes whereas rules for static attributes can only depend upon other static attributes.

2.2 Reconfiguration Modes

An important feature of the ARC system is the ability to do performance modeling with reconfiguration of the target architecture. Each module, carrier, and port can have associated with it a *configuration mode*. This mode defines the various configurations particular components can have. When a module, carrier, or port is declared with a configuration mode, a builtin attribute called *mode* is automatically attached to the component. The data type of the *mode* attribute is defined in the model specification and can be referenced in any evaluation rule. Thus, evaluation rules can be formulated that calculate different values for a specific attribute based upon the particular mode of a certain component.

2.3 Reconfiguration Triggers

Another builtin attribute is the *trigger* attribute. This attribute is a boolean type. It is only created for modules, carriers, or ports that are defined with a configuration mode. This attribute is slightly different than others because it is used to govern the evaluation semantics of the ARC system during evaluation.

It is an attribute that governs when a module, carrier, or port will change its configuration. When this attribute becomes true, the mode of the particular design component is changed to the next corresponding mode. The next configuration mode can be defined by an evaluation rule or assigned by the scheduler. Effectively, if there is no evaluation rule defining the next mode, the application will define the next configuration mode. There are several mechanisms that can be employed by the application to do this via the application procedural interface.

3 Reconfigurable Architecture Specification

The architecture specification is a description of the actual components of the target architecture. Recall that the performance model description has declarations for the various components that will be in the architecture specification. Thus, the architecture specification will contain instances of each of those component types along with any hierarchy and connectivity information that may be required.

4 Composer

The composer creates an executable performance model using the performance model specification and architecture specification. The performance model specification without an architecture specification is not executable and can not be

used to perform any kind of calculation. Composing an executable performance model involves several steps [7].

In the first step, each component from the architecture specification is taken and augmented with the attributes defined for that component in the model specification. After all attributes are attached to the components, the performance model, which is merely a collection of the various evaluation rules, is created.

Once the executable model is created, several analysis steps are performed to ensure that the model can be evaluated. First a dependency graph representing the dependencies among the various evaluation rules is created for the entire performance model. That is, each evaluation rule is analyzed to determine those attributes upon which it depends and correspondingly represented in the graph. Then the dependency graph is searched for cycles and topologically sorted to determine an evaluation sequence for the evaluation rules. With cycles in the graph, it is not possible to determine such an evaluation order [8]. Thus, the composer reports such cycles and details which attributes are involved in various cycles.

5 Evaluator

Input to the evaluator is the sorted dependency graph. Evaluation involves starting with all attributes of order 1 and obtaining values for those attributes. Such attributes must be assigned a value from some external source via the API. A user can directly assign values through a command line shell or they can be assigned by other tools or applications.

Once the attributes of order 1 have been assigned values, the rest of the attributes are evaluated in the correct order. Results of the evaluation are stored in a results database. A results database is maintained for all attributes that are ever evaluated, even between various reconfigurations. It is possible that in different configurations there may several attributes which do not exist in other configurations. The database maintains a history of all attributes and their corresponding evaluation results for each of the various configurations.

6 Scheduler

The main function of the scheduler is to handle the reconfigurations of the performance model. The scheduler is responsible for controlling and responding to the different *trigger* attributes within the design components.

When triggered to do so, the scheduler is responsible for assigning the new configuration modes of the various design components and ensuring that the dependency graph is updated with any changes caused by moving to a new configuration. Thus, it is tightly coupled with the composer and evaluator to perform the necessary operations and updates for reconfiguration.

7 Triggers and Applications

One possible controlling input to the ARC system that makes it flexible for a variety of target architectures is an application behavior. ARC views an application as a series of states, where each state represents a particular configuration of the application. The application behavior can be defined in C or C++. The application can control the ARC system by using the triggers and configuration modes. From within the application behavior, at various transitions from state to state, an application can use the triggers and configuration modes to change the performance model as required.

Configurations modes can be used at various levels of abstraction depending upon the architecture and performance model specification. Thus, configuration modes do not only apply to low level bitmap type configurations.

8 Conclusion

The ARC System is an ongoing project that is currently under development. For more specific details of the ideas and concepts presented in this paper, see the technical report entitled "ARC Technical Report #980123" on the ARC home page via the URL http://www.ececs.uc.edu/~ddel under the research projects link.

References

1. C. Iseli and E. Sanchez. A C++ Compiler for FPGA Custom Units Synthesis. In *IEEE Conference on FPGAs for Custom Computing Machines*, pages 173–179, 1995.
2. Michael J. Wirthlin and Brad L. Hutchings. A Dynamic Instruction Set Computer. In *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 99–107, 1995.
3. Ranga Vemuri, Ram Mandayam, Vijay Meduri. Performance Modeling Using PDL. *IEEE Computer*, pages 44–53, April 1996.
4. Jeffrey Walrath and Ranga Vemuri. Symbolic Evaluation of Performance Models for Tradeoff Visualization. In *Proceedings of 34th Design Automation Conference*, pages 359–364, 1997.
5. M. Nagl and A. Schurr. A Specification Environment for Graph Grammars. In H. Ehrig and H.-J. Kreowski, editors, *Graph Grammars and their Application to Computer Science*, volume 411, pages 599–609. Springer-Verlag, 1991.
6. U. Kastens. Ordered Attribute Grammars. In *Acta Informatica*, volume 13, pages 229–256, 1980.
7. K. Kennedy and S.K. Warren. Automatic Generation of Efficient Evaluators for Attribute Grammars. In *Proceeding of 3rd ACM Symposium on Principles of Programming Languages*, pages 32–49, 1976.
8. M. Jazayeri, W. Ogden, and W. Rounds. The Intrinsically Exponential Complexity of the Circularity Problem for Attribute Grammars. *Communications of the ACM*, 18:679–706, December 1975.