X-Sim: A Federated Heterogeneous Simulation Environment

Saurabh Gayen
Eric J. Tyson
Mark A. Franklin
Roger D. Chamberlain
Patrick Crowley

Saurabh Gayen, Eric J. Tyson, Mark A. Franklin, Roger D. Chamberlain, Patrick Crowley, "X-Sim: A Federated Heterogeneous Simulation Environment," in *Proc. of 10th High Performance Embedded Computing Workshop*, September 2006, pp. 75-76.

Dept. of Computer Science and Engineering
Washington University
Campus Box 1045
One Brookings Dr.
St. Louis, MO 63130-4899

X-Sim: A Federated Heterogeneous Simulation Environment

Saurabh Gayen, Eric J. Tyson, Mark A. Franklin, Roger D. Chamberlain, Patrick Crowley
Department of Computer Science and Engineering
Washington University in St. Louis
sg3@wustl.edu, etyson@wustl.edu, jbf@wustl.edu, roger@wustl.edu, pcrowley@wustl.edu

Introduction

Heterogeneous systems consisting of a multitude of different platforms are often the target for high-performance applications, either due to a desire to leverage the unique strengths of each platform, or simply to use the devices that are readily available. Developing applications to run on such a diverse set of devices, however, is a daunting task.

Coordinating multiple languages, especially very different ones like hardware and software languages, is awkward and error-prone. Additionally, implementing communication mechanisms between different devices unnecessarily increases development time. Simulating such a system is complicated because of the need to coordinate compilers and simulators, often with very different interfaces, options, and fidelities.

Auto-pipe [1] is a toolset developed to address these difficulties. Auto-Pipe applications are expressed using a data flow coordination language. The applications may then be compiled and mapped onto complex sets of devices, simulated, and optimized.

X-Sim is a simulation environment that is part of the larger Auto-Pipe system. It is used to establish functional correctness and gather performance statistics. Key features of X-Sim are:

- Integration of multiple, potentially very different simulators, into a single federated simulation.
- Automation of the system simulation by coordinating individual simulator runs.
- Integration into the Auto-Pipe design flow.

The Auto-Pipe System

Auto-Pipe is a performance-oriented development environment for heterogeneous systems. It concentrates on streaming applications placed on pipelined architectures. In this system, applications are expressed in the X Language [2] as dataflow graphs. In these graphs, individual computational tasks called *blocks* are connected with interconnections called *edges* indicating the type and flow of data between blocks. The actual implementations of the blocks are written in various languages for any subset of the available platforms (e.g., C/C++ for general-purpose processors, HDL for FPGAs, assembly language for network processors and DSPs).

This work is supported by NSF grant CCF-0427794.

Applications may be mapped onto arbitrary sets of mixed physical resources. Such resources may be a combination of general-purpose processors (e.g., x86, PowerPC, ARM), chip multiprocessors (e.g., Intel IXP network processor, multi-core x86 processors, IBM Cell processor), and reconfigurable hardware devices (e.g., FPGAs).

The Auto-Pipe environment consists of a set of tools for compiling, simulating, and optimizing applications in complex heterogeneous systems. These tools include X-Com, an X Language compiler; X-Sim, a simulation environment; and X-Opt, an optimization tool. Auto-Pipe aims to provide an extensible infrastructure for supporting new types of computation and interconnection devices, simulators, and optimization techniques.

Within the Auto-Pipe system, X-Sim provides *functional simulation* to determine application correctness, and *performance simulation* to profile individual components of the application. The results of performance simulation may then be used to improve the performance either manually or automatically using the X-Opt tool.

Figure 1 depicts an Auto-Pipe design flow which automatically optimizes mapping of an application's blocks and edges onto physical computation resources (e.g., processors, FPGAs) and interconnect resources (e.g., PCI bus, Ethernet switch). The mapped application is first compiled using X-Com. Then, X-Sim simulates the compiled components and generates performance statistics. After checking for functional correctness, X-Opt uses the statistics to incrementally revise the mapping.

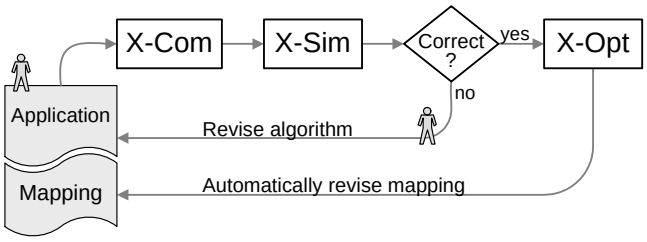

Figure 1: Design flow under Auto-Pipe

The X-Sim Environment

X-Sim provides an environment in which multiple simulators are seamlessly combined to simulate X Language applications mapped to heterogeneous devices. The X-Sim infrastructure is open-ended to allow support for a range of simulators, from low-level discrete-event and cycle-accurate simulators to rough estimates from analytic models. X-Sim supports the SimpleScalar [3] processor simulator, the ModelSim [4] hardware logic simulator, and native execution in POSIX environments.

X-Sim generates Makefile scripts to automatically compile and simulate applications. The scripts use data dependencies derived from the application description to simulate individual components in order. In acyclic pipelined systems, this allows one to run simulations concurrently on multiple machines.

Interconnects between computation devices are simulated by X-Sim using an extensible library of communication models. If a more complex communication simulator is available, that may be used instead.

Figure 2 shows how an application with four blocks (A, B, C, D) distributed across two devices looks when simulated with X-Sim. Directed arrows depict the flow of data, with inter-device communication using data files, and in-device edges using the native communication methods (wires in FPGA, function calls on a processor). To profile application performance, X-Sim keeps track of when data enters and exits individual simulators by maintaining multiple timestamp files (T1, T2, T3) for every interconnect. Interconnect models are used on all inter-device communications to simulate data transmission.

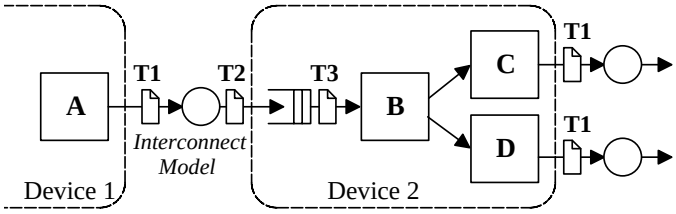

Figure 2: Flow of data within an Auto-Pipe simulation

There are three types of timestamps. The first timestamp (T1 in Figure 2) keeps track of when data is output from a computational device onto an interconnect. The second timestamp (T2) indicates when the data has been transmitted across the interconnect and is available to the receiving device. The third timestamp (T3) records the time at which the receiving device consumed the data and started processing it. By maintaining these timestamps, X-Sim provides a time trace of all data transfers that occurred between computational devices.

Multiple blocks may be mapped to the same computational resource. Timestamps are only kept for the data entering and exiting blocks which connect to interconnects.

When they are executed, each simulator generates entry and exit timestamps in their native format. Header information indicates in what format the timestamps are stored. After all the timestamps are collected, this format information is used to normalize them into a universal time domain. The timestamps may then be used to generate characteristic distributions of processing times for each device.

An analysis component obtains basic and advanced performance measurements using the timestamps. Basic measurements include the mean and variance of service time distributions associated with devices. These measurements may be easily aggregated to determine throughput and latency figures for the individual devices and the system as a whole.

More advanced measurements may take the time distribution data and fit it to common analytic distributions (e.g., exponential, gamma, Gaussian) for the development of queuing models. The raw data may also be used in trace-driven simulations.

A user may take these simple and complex performance analyses and solve for various performance parameters using analytic methods. Using this capability, alternative mappings may be explored to improve performance, either manually by user inspection, or automatically by using the X-Opt tool.

Currently, X-Sim requires that the devices in a simulation be organized in an acyclic pipeline. This is because all intermediate data and timestamp files must be completed before they may be used by the next stage.

An advantage of the X-Sim approach is that the simulator data dependencies are well understood a priori. This allows independent simulations to run in parallel. By exposing the data dependencies in the simulation Makefile and using the file system for communication, X-Sim is able to easily distribute simulations of large systems across many real computing resources to speed up simulation.

Conclusions

We have introduced X-Sim, a simulation environment for the Auto-Pipe toolset, and described how it functions. X-Sim supports the infrastructure of Auto-Pipe to simulate X applications distributed onto complex heterogeneous architectures. X-Sim is integral to the Auto-Pipe design loop involving automatic improvement of pipeline performance.

Future work may investigate permitting cyclic topologies when all simulators in the system permit the pausing of simulation time, which will allow the entire distributed system to run incrementally.

X-Sim currently supports the ModelSim and SimpleScalar simulators for detailed hardware and software simulation. It also supports native execution for performance estimates of off-the-shelf computer systems. Furthermore, X-Sim is easily extended to support more simulator and emulator environments and performance models as more device types and development platforms are integrated into the Auto-Pipe toolset.

References

[1] Mark A. Franklin, Eric J. Tyson, James Buckley, Patrick Crowley, and John Maschmeyer. “Auto-Pipe and the X Language: A Pipeline Design Tool and Description Language.” In *Proc. of Int’l Parallel and Distributed Processing Symp.*, April 2006.

[2] Eric Tyson. *X Language specification v1.0*. Technical report WUCSE-2005-47, Washington University Dept. of Computer Science and Engineering, 2005.

[3] Todd Austin, Eric Larson, and Dan Ernst. “SimpleScalar: An Infrastructure for Computer System Modeling.” *Computer*, vol. 35, no. 2, pp. 59-67, Feb. 2002.

[4] Mentor Graphics Corp. ModelSim. http://www.model.com