Rapid Prototyping and Emulation of Many-Core Chip Multiprocessors with Integrated Hardware Accelerators

I. MOTIVATION

Simulation of many-core systems is becoming an increasingly restrictive bottleneck because current simulators do not scale well to a large number of processors [1]. Current full system simulators such as Virtutech’s Simics [2] incur a slowdown of 3-4 orders of magnitude [3]. This results in over a month of simulation in order to run the SPEC2k benchmark suite [4], [5]. As more cores are added to a system, this slowdown increases exponentially [3].

Hardware emulation, an alternative to simulation, is proposed to test and evaluate new architectures. FPGAs are a natural fit as an emulation platform because they contain a flexible interconnect and a sea of logic for implementing various components.

In this work the Architecture Compiler for Model Emulation (ACME) system, in conjunction with the SuperCISC compiler, is used to develop several hardware accelerated systems. ACME is a tool flow designed for rapid prototyping and emulation of large systems. In particular, ACME takes a graphical description of a design and automatically generates a system-on-a-chip model to emulate the system using an FPGA. The SuperCISC compiler is a C to VHDL hardware acceleration tool flow. ACME and the SuperCISC compiler are described in detail in [6] and [7] respectively.

II. RELATED WORK

A great deal of research effort has been placed into addressing the issue of simulating large many-core systems. In [5] and [4] the SMARTS framework attempts to reduce simulation times by only sampling various simulation points of execution. Once a simulation point is reached, a detailed simulation is performed for a specified time window. In [8] this technique is improved upon by analyzing the program first in order to determine simulation points which are most representative of program behavior.

The Research Accelerator for Multiple Processors (RAMP) project is a multi-university project aimed at creating new computer architecture research tools by using FPGAs to emulate parallel computer systems [9]. The RAMP Blue is a prototype system for multi-core emulation using FPGAs [10]. Unlike RAMP, ACME provides a high-level design environment that avoids the need to develop in cumbersome hardware languages.

III. APPROACH

A high level view of ACME is shown in Figure 1 and will be referenced throughout the text. In the figure, lightly shaded regions are provided by various third parties and the darker segments of the flow were developed by the author.

**Fig. 1.** High level view of the ACME system-on-a-chip tool flow.

*Ptolemy II* is a graphical entry tool from UC Berkeley used for modeling systems [11], [12]. Basic hardware components such as multiplexers are provided in Ptolemy’s *Java Actor Library*. These components are used to create larger composite models.

The ACME tool starts with a library of VHDL implementations of the relevant Ptolemy actors shown in *VHDL Actor Library*. Based on this library the *ACME Front End* can read the composite model and generate a synthesizable VHDL description of the model which can be run on a FPGA. The *ACME Xilinx Back End* takes the SoC model and VHDL from the front end and generates a Xilinx Platform Studio project that instantiates Microblaze processors and the appropriate code to interface with the VHDL [13].

In order to more quickly design complex components, ACME also provides the ability to describe functionality in C code. This C code is executed on FPGA soft-core processors, but can also interface with Ptolemy via the Java Native Interface [14]. These processor based actors give ACME the ability to easily generate large MPSoC designs. The SuperCISC compiler can also be used to turn all or part of the C code into custom logic. This allows the user to write C code for simplicity while still capitalizing on the performance gains of hardware acceleration.

The resulting system works synchronously, where each component proceeds according to an emulated *target clock*, which is different from the *host clock* of the FPGA. Differentiating between the two allows for the synthesis of timing constructs, as well as more effective decoupling of the design from the host FPGA. A hardware barrier, based on the circuit introduced in [15], is used to ensure that the processors and logic remain in sync with the target clock in order to generate cycle-accurate results.

ACME also allows the user to specify components’ latency
and throughput via parameters in Ptolemy. This allows multiple copies of a component to be tested simultaneously with different timing characteristics.

IV. RESULTS

ACME was compared to Simics and Ptolemy by testing a 16-core and a 64-core packet switching mesh network. The systems were tested by running the Raytrace benchmark from the SPLASH-2 multi-threaded benchmark suite [16]. The 16-core (4x4) mesh executed a 1,000 message segment of the application, while the 64-core (8x8) mesh executed 5,000 messages. The runtimes and speedups from executing the network simulation are shown in Table I. The Ptolemy and Simics simulations were run on an Intel Xeon 8 core machine operating at 2.3 GHz with 8 GB of RAM. The emulation was executed on a Xilinx Virtex 5 110T operating at 125 MHz. The emulation system ran four times as fast as Simics for 64-cores, indicating a super-linear slowdown in Simics.

The emulation system ran four times as fast as Simics for the 16-core experiments. This speedup increases to over 23x for 64-cores, indicating a super-linear slowdown in Simics. Unlike the hardware solution, an event driven system like Simics slows considerably, creating limits on the number of cores that can be considered. While the emulation system outperforms Simics and Ptolemy, it is possible to achieve even greater speedups. A major performance bottleneck in the mesh arises from the processors running switch arbitration code. Depending on network traffic, a single target cycle of arbitration can take upwards of 450 native cycles. To study the effects of hardware acceleration on computationally intensive C code, several image and signal processing benchmarks were studied within the ACME framework. Figure 2 shows the speedups achieved by using the SuperCISC compiler to accelerate the benchmarks’ execution kernels. The results show that, on average, an additional speedup of 2.32x is achieved. The SuperCISC compiler was then used to accelerate the primary code used for arbitration in the mesh network. The arbiter code speedup was over 3x. This led to overall speedups of 14x and 73x for the 16 and 64-core meshes.

V. CONTACT INFORMATION

Colin J. Ihrig,
Graduate Student,
ACM SIGDA Student Member: 8880964
cji3@pitt.edu

348 Benedum Hall
Computer Engineering Program
University of Pittsburgh
Pittsburgh, PA, 15261

Advised by: Alex K. Jones

REFERENCES