A Flexible Framework for Communication Evaluation in SoC Design

Praveen Kalla
University of Notre Dame
nkalla@cse.nd.edu

Xiaobo Sharon Hu
University of Notre Dame
shu@cse.nd.edu

Jörg Henkel
University of Karlsruhe
henkel@informatik.uni-karlsruhe.de

***Abstract*—**
We present SoCExplore, a framework for fast communication-centric design space exploration of complex SoCs with network-based interconnects. Speed-up in exploration is achieved through abstraction of computation as a high-level trace, and accuracy is maintained through cycle-accurate interconnect simulation. The flexibility offered allows for fast partition/mapping and interconnect design space exploration. Error analysis of such frameworks is non-trivial and is presented for the first time. As a case study, a speed-up of 94% over architectural simulation is reported for the MPEG application.

I. INTRODUCTION

Future SoCs with 1 billion transistors will comprise hundreds or even thousands of heterogeneous multiprocessors on a single chip. In these systems, communication will become a major concern since it eventually determines whether or not the computation resources can efficiently be deployed i.e. communicate with each other. In recent years the design paradigm of Networks-on-Chips (NoC) has evolved as a possible solution ([2] [4]).

One of the challenges of introducing networks on chips is to efficiently evaluate different communication architectures. To achieve high efficiency, works like [3], [6] and [13] employ statistical models to generate interconnect traffic and conduct necessary analysis based on the statistically distributed packets. Such frameworks might not be able to capture true communication behavior in an application. To overcome this problem, simulation tools that resort to a complete execution model have been proposed (e.g., [1] [8]). Here, an application is re-written to make communication explicit, and encapsulated in a system-level language such as SystemC. The demands on understanding and re-writing application code together with the detailed simulation can make the design space exploration rather inefficient. Trace-based/compiled-code simulation approaches (e.g., [9]) can significantly speedup the design space exploration process. Existing frameworks, however, assume that the task execution order of an application does not change according to interconnect behavior (e.g, [7] [11] [14]). In such cases, even though the functional behavior of the application is captured, *the true execution order is not*.

Consider the example in Fig. 1(a) where two tasks (t_1, t_2) are assigned to a single core $(c1)$ in a multiprocessor system. t_1 has three computation blocks (t_{11}, t_{12}, t_{13}) with data transactions d_1 and d_2 after t_{11} and t_{12} respectively. t_2 has the computation blocks t_{21}, t_{22}, t_{23} with data transactions d_3 and d_4. Say, t_2 has been constrained to execute after t_1 even though t_1 and t_2 are independent processes. The time line for the system is shown in Fig. 1(b). The gaps indicate core-idle time due to interconnect delay in satisfying the data transactions. However, in a real system where the cores would context switch, the time-line is as shown in Fig. 1(c). The task execution order, communication event profile and total execution time are all affected due to constrained execution.

In this paper, we overcome the shortcomings of previous trace-based simulation through **two contributions:** (1) a flexible communication event simulator that does not require task execution order be given and that can take traces at different levels of abstraction. (2) a detailed error analysis of trace-driven simulators based on a novel *bounded interconnect access model* and synchronization points. The effectiveness of the framework is demonstrated through a case study.

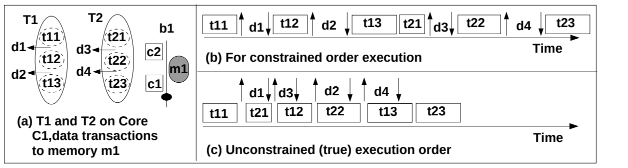
T1
T2
b1
d1
d2
d3
d4
t11
t12
t13
t21
t22
t23
c2
c1
m1
(a) T1 and T2 on Core C1,data transactions to memory m1
(b) For constrained order execution
(c) Unconstrained (true) execution order
Time

Fig. 1. Effect of constrained execution order

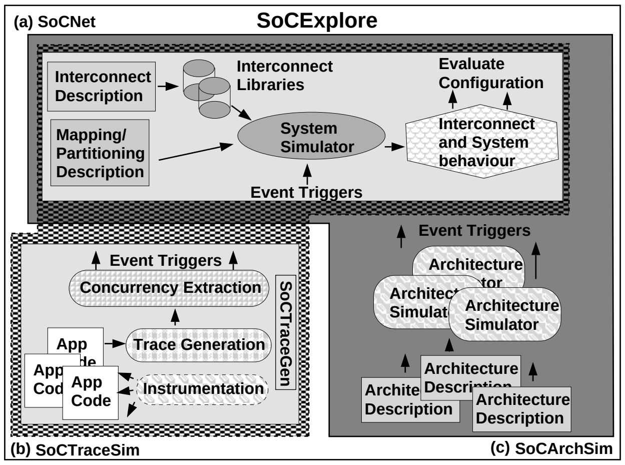

Fig. 2. The SoCExplore Framework

II. SoCExplore

We present our framework, SoCExplore in Fig. 2. The central component is the event-driven cycle-accurate interconnect simulator, **SoCNet** (Fig. 2(a)). Interconnect and other system events in SoCNet are triggered by hooks to higher level mechanisms. Next we propose **SoCTraceSim**, a trace-based system simulation platform that incorporates SoCNet and is based on the use of *concurrent execution traces*. An original application trace (interconnect transaction triggers interspersed with computation events) is modified to extract concurrency(parallelism) in **SoCTraceGen** (Fig. 2(b)). This extraction separates application space from system design space to a great extent and thus multiple interconnect/partition/mapping configurations; and systems with multiple copies of the applications, can be explored through multiple runs of SoCNet without involving trace-regeneration. Interconnect events directly influence the release of computation events through a feedback mechanism and thus task execution order is maintained close to that in a real system. Trace-based frameworks abstract computation and trade accuracy for speed and induce errors in communication-event release times. As such, we built a reference platform, **SoCArchSim** (Fig. 2(c))(which uses SoCNet for cycle-accurate communication and architectural simulators for computation simulation) for design space exploration at later stages as well as to quantify the error in SoCTraceSim when needed. In the following we describe application modeling in SoCExplore followed by details of its various components.

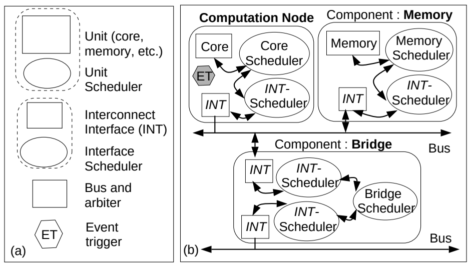

Fig. 3. (a) Software Architecture for system component (b) Typical System

A. Modeling in SoCExplore

An application is described by **A(T, D, R)** where
T : the set of tasks t_i (assuming that the high-level specification is given as a C/C++ code, t_i represent the functions in the code to be mapped as a whole to computation nodes).
D : the set of data variables d_i, accessed by **T**.
R : the control relation among t_i's, e.g., (t_1 *calls* t_2).
The SoC system is represented as **S(C, M, I)**, where **C** is the set of concurrently executing cores (or computation nodes), c_i; **M** is the set of memory units, m_i; and **I** is the communication architecture describing the interconnect elements such as buses, arbiters, routers as well as the protocols that define their functionality, such as interconnect-access model, request-scheduling at routers/arbiters. The goal of communication-centric design space exploration is to find superior mappings ($T \Rightarrow C, D \Rightarrow M$) in terms of communication cost (e.g., buffer sizes) and/or performance (e.g., total cycles needed for data transfer). Next we discuss the details of the interconnect simulator, SoCNet.

B. SoCNet: Interconnect Simulator for On-Chip Networks

SoCNet evaluates the performance of an application A(T, D, R) on a system S(C, M, I) by simulating its communication behavior. The inputs to SoCNet are the system configuration file (SF), a user-defined mapping granularity file (UDGF), and a high-level concurrent execution mechanism (HCEM), e.g. a multi-processor architectural simulator; or its abstraction, ACEM, such as a concurrent execution trace. We discuss HCEM and ACEM later.

Each component of S(C, M, I) is described in SoCNet as a composition of some basic entities shown in Fig. 3(a). <*unit*>_schedulers order the events in a <*unit*> (execution core, memory unit, etc.) and <*INT*>_schedulers order the data transactions at the corresponding interconnect interface, <*INT*> (please refer to [5] for a detailed description). These entities are described at the needed complexity to achieve cycle-level accuracy for interconnect events. A sample system is shown in Fig. 3(b).

SoCNet is closely coupled to a HCEM/ACEM through *event-triggers*(the ET entity). The ETs in the computation nodes pass triggers (<*task_start*>, <*task_resume*>) from HCEM/ACEM to the execution cores. When these tasks generate interconnect transactions, a <*context_switch*> feedback-trigger is passed on to the HCEM/ACEM through the ET. Also when interconnect transactions are satisfied, a <*transaction_complete*> feedback is passed on to the HCEM/ACEM. The interconnect events are simulated at cycle-level accuracy whereas that of task execution can be controlled by the user to achieve accuracy (HCEM) or speed and flexibility (ACEM). These are described in the following sections.

C. SoCTraceGen: High-Level Trace Generator

The goal of employing ACEM is to speedup design space exploration through 1) abstraction of computation complexity into *events* and 2) concurrency extraction to reduce application-to-system mapping complexity. SoCTraceGen accomplishes these tasks with the help of UDGF through the following phases:

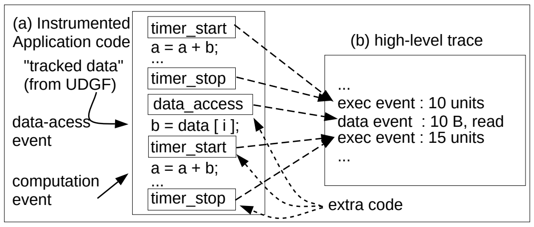

Fig. 4. Transformation of trace

High-level trace generation : The UDGF contains information regarding shared data elements among tasks as well as those that might generate interconnect transactions under a mapping (these data elements might be mapped to non-local storage structures). Based on this information, the application code is manually instrumented as shown in Fig. 4 and executed on a host machine. Computation between accesses to the “tracked” data, d_{tr}, is abstracted along with the details of the data access into a high-level trace.

Concurrency extraction : To support rapid exploration of system configurations, application space (C/C++ code description) has to be separated from system space so that a code re-write is not necessary for changes in mapping (T ⇒ C, D ⇒ M) or components (number of cores, memory units, etc). This separation is achieved by extracting concurrency from the application.

Concurrencies among the tasks are determined by d_{tr}/control dependencies. Tasks are annotated with multiple suspend/resume points similar to context-switch points in a process in a multi-tasking environment. These points correspond to function calls/returns/non-local d_{tr} accesses. We refer to segments of code between suspend/resume points as *task segments*. Extracting temporal concurrency involves identifying the instances of task segments (belonging to different tasks) that can be executed concurrently if allowed by a mapping. This bears similarities with what is done in parallel compilers or extracting data/control flow graphs from sequential code. The difference is that the extraction is carried out on an *execution trace* rather than on a high-level code.(we skip the details due to the page limit)

As can be guessed from the above description, there can be several factors that might contribute to an error in the input trace to SoCNet which are discussed in later sections. As such, the SoCExplore framework also incorporates a lower abstraction level approach, which is described next.

D. SoCArchSim: Micro-architectural System Simulation

SoCArchSim (Fig. 2(c)) is a multiprocessor micro-architectural simulator that interacts with SoCNet. It can be used in later stages for design refinement and can also provide a reference for evaluating ACEM based approaches. It consists of a set of cycle-accurate architecture simulators([12]) that are hooked up to SoCNet to generate event-triggers. Simulation proceeds similar to the case when high-level traces are used, except that computation events are now *cycle-accurate*. Unlike the SoCTraceGen approach, no extra code is embedded into the application. Instruction and data addresses are monitored as they are put on the address bus of the architecture simulator and appropriate events are generated when “tracked” tasks or data are accessed. Thus computation and data access abstractions are accurate.

III. Error Analysis

In this section we discuss various sources of errors in SoCExplore and their bounds (if any). An “error” in this context means the following: the deviation between the output of SoCTraceSim (referred to as trace-based behavior) and of SoCArchSim (referred to as “true” behavior).

A. Error Sources

We categorize errors of trace-based behavior into two types:
a) **Input-trace error :** The input trace is an abstraction of *computation events* and *data transactions* on the interconnect, generated from a *particular* input-data set. There can be errors in both types of events if the architectures of the target core (in SoCNet) and that of the host (where the trace is generated) do not match. An error in computation events is due to factors such as in-order/out-of-order execution, instruction fetch/issue bandwidths, cache sizes and hierarchy. An error in data transactions is typically due to the size of store-forward queues and speculative fetches. The former hides certain transactions from coming out onto the interconnect while the later might result in extra transactions. Instrumentation code might further disturb normal execution and add to the error. Input-data variations also induce input-trace error. Please refer to [5] for detailed experimental data on the above issues. We denote the input-trace error as $e_{ip} = (e_{comp}, e_{data})$, where e_{comp} and e_{data} can be defined at different complexities such as per-event, per-task-instance, etc.
b) **Simulation error :** This error occurs during the simulation process of SoCNet due to execution order perturbation. Error in abstraction of computation events can perturb the release/propagation of interconnect events. This in turn perturbs the choice/release of the next computation event to be scheduled on the execution cores. We denote this error as e_{ord}. The error due to the concurrency extraction phase of SoCTraceGen can be minimized with a well-defined UDGF.

A framework is useful only if error bounds can be defined and are non-trivial (maximum or minimum possible activity). In the following, we investigate the impact of a given e_{ip}, and e_{ord} on the output data.

B. Impact of Trace Error and Interconnect Access Models

Here we examine the two components of e_{ip}: e_{comp} and e_{data}, respectively. The error e_{comp} impacts the timing data of a computation event in the trace, which in turn impacts the timing data of the output. Since we have isolated the effect of errors in timing data on the execution order as a simulation error; we only need to focus on how e_{comp} impacts the total execution latency of the system. In the worst case, e_{comp} for each computation event on each core is accumulated (which is similar to the effect observed in a system with a single execution core).

The effect of e_{data} is more complex. Each data transaction generates certain interconnect events based on the *access model* of the interconnect. An error in a data transaction causes additional or reduced number of interconnect events. If we can bound the interconnect events generated per data transaction by B_i, the total error in interconnect *events* due to e_{data} is bounded by $e_{data} \cdot B_i$. The error bound in system *latency* due to e_{data} can be determined similarly.

As to whether B_i exists or not in the first place, depends on the *interconnect access model*. Consider a system where all components on a bus have limited receive buffers (*rxbuf*). Consider the following access models:
AM1: When the *rxbuf* at the destination is full, the source attempts to query the receiver continuously till the transaction is resumed.
AM2: The state of the *rxbuf* at each component is maintained at every other component and a source transmits only when acceptable by the receiver.

Now consider a transaction where a core writes a large data packet to memory which requires several interconnect transfers. If the core and the memory are on a single bus, AM1 and AM2 result in a fixed number of interconnect events with AM1 generating extra query-events. Now consider a more complex case where this transaction has to travel through multiple buses with buffers at each router along the path. Assume that all buffers are full and are waiting for the memory to complete a write cycle. The query-transactions will show a domino effect and hence each data transaction can be expected to generate maximum workload. e_{data} can be magnified “infinitely” under model AM1 in such transactions. However, consider AM2 which does not have any redundant query transactions. Here the activity is bounded per data transaction and hence a non-trivial error bound does exist.

C. Impact of Execution-Order Error and Observation Points

The execution-order error, e_{ord}, which is induced by the input-trace error, does not affect the *total* system latency or the *total* interconnect activity since it does not alter the total number of transactions (under access models like AM2 in the previous section). However, its effect becomes non-negligible if the designer wishes to observe the instantaneous interconnect/system state. As such, trace-based simulation results would not provide reliable information except at the end of simulation.

The reason why a (high-level) trace-based simulation cannot bound the error at instants other than the end of simulation is due to the absence of “synchronization” information which would otherwise establish a deterministic state in a “true” system (a system architecture simulator such as SoCArchSim) at a certain instant. This limitation can be relaxed for many applications where one can actually define some intermediate synchronization points. Consider a system that receives input data frames at a periodic rate and has a hard deadline for processing these frames, i.e., the system has to process a data frame before the next one arrives. Let the arrival of a frame be indicated by an interrupt. This system inherently has a deterministic state, (i.e.“system idle”), at the instant before the interrupt occurs. In such a context, traced-based simulation is capable of providing instantaneous interconnect state at instants when the interrupt occurs. In the general case, assume that N applications with input data frame rates, R_i for $i = 1, 2, \cdots, N$, are executed on a system. There exists a point $R_{LCM} = LCM(R_1, R_2, \cdots, R_N)$, where LCM is the least common multiple, at which instant all applications have completed processing and the system is at an instantaneous “idle-state”. Trace-based simulation can provide reliable information about the total activity during the last R_{LCM} with well-defined error bounds which can be computed as described in [5]. We illustrate the above developed concepts with a case study in the next section.

IV. Case Study for the MPEG Application

The case study for the MPEG application([10]) is summarized in Fig. 5. Six tasks and twelve data variables are used in the UDGF for a sample system consisting of five execution cores and four memory units. The different interconnect topologies considered are as shown. Multiple (three in our case) copies of MPEG are taken as the base application since this might be typical of future SoCs.

To illustrate the flexibility and speedup of SoCNet in the case of load balancing in a system, we present the following: Tab. I illustrates how the load on different components can be balanced by using different mappings. It presents the utilization of each component in Config. 5 (Fig. 5). Concentrating on the results from SoCTraceSim, it can be seen that M2 has

TABLE I

LOAD BALANCING AND ACCURACY RESULTS USING SOCEXPLORE FOR CONFIG. 5 (REF. FIG. 5)

Mapping	Framework	Total Cycles	Core Utilization					Memory Utilization				Bus Utilization					
			c0	c1	c2	c3	c4	m1	m2	m3	m4	b0	b1	b2	b3	b4	b5
Map 1	SoCTraceSim	28,497,339	0.22	0.33	0.39	0.19	0.12	0.00	0.00	0.50	0.07	0.25	0.07	0.16	0.42	0.07	0.15
	SoCArchSim	26,679,673	0.15	0.21	0.25	0.15	0.13	0.0	0.00	0.49	0.06	0.25	0.08	0.16	0.43	0.06	0.14
Map 2	SoCTraceSim	**23,780,310**	**0.34**	**0.15**	0.34	0.15	**0.49**	**0.20**	**0.23**	**0.22**	0.03	0.20	**0.28**	0.12	**0.23**	**0.23**	**0.25**
	SoCArchSim	21,686,957	0.23	0.13	0.23	0.13	0.37	0.18	0.21	0.21	0.03	0.20	0.29	0.12	0.23	0.23	0.27

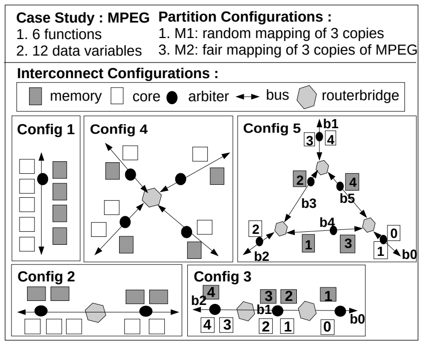

Fig. 5. Design space exploration for MPEG

TABLE II

PERFORMANCE OF DIFFERENT FRAMEWORKS

Framework	Phases	Time (1st run)	Total Time (for 20 configs)
SoCTraceSim	Trace Generation	7 min	
	Concurrency Extraction	$<$ 1 min	
	SoCNet	3 min	68 min.
	Total (for initial run)	11 min	**(94% speedup)**
SoCArchSim	for each configuration	52 min	1040 min.

spread the core/memory and bus utilization more evenly (e.g., highlighted values in Tab. I). (We omit detailed results on a per-component basis such as temporal behavior, etc., for lack of space). Consider the computation expense involved in trace-based and architectural simulation as shown in Tab. II. A 94% speedup is achieved using SoCTraceSim. The obtained speed-up comes at a slight loss of accuracy as noticed from the results of SoCArchSim shown in Tab. I. The errors are due to the many factors as explained in Sec. III.

As a detailed example, refer to Fig. 6 which depicts the bus activity (total clocks for which the bus was active) for bus 1 in Config. 5 in Fig. 5. It can be seen that the errors are quite small. However one may not trust these results unless error bounds are established. Using the equations developed in [5], the calculated error bounds are presented in Fig. 6.

Note that the upper bound has a linear rise corresponding to maximum possible activity and then flattens out once it reaches the maximum. The previous analysis was for a run where each MPEG application processed 5 frames of data. As can be seen, the reliability of the framework is best at the end of the simulation, i.e, parameters like total energy and run-time can be specified more accurately at end-point of the run, but not at other instants (greater error).

As described in Sec. III-C, synchronization points, however, increase the reliability of the results from the framework. Fig. 7 illustrates the significance of synchronization points. The system is the same as before with 3 copies of MPEG application. Only now, frames are delivered to the system at particular points (after the previous frames finish) and an interrupt is generated. The error bounds at the end of the frame and during a frame are presented. Note the increase in reliability through the application run. Also note that the bounds diverge as the number of frames increase. This is due to the cumulative effect of the error from previous frames.

As a conclusion, it can be seen that error analysis for trace-based communication-centric exploration frameworks such as SoCExplore is non-trivial. However the speedup (over 94%)

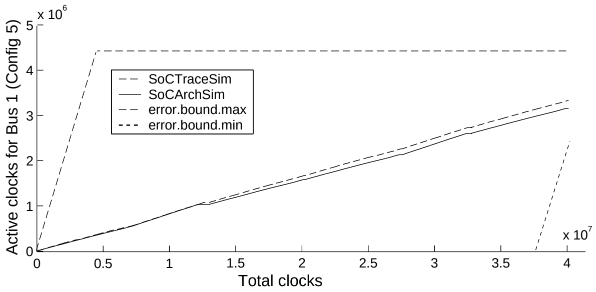

Fig. 6. Error bounds for un-synchronized system

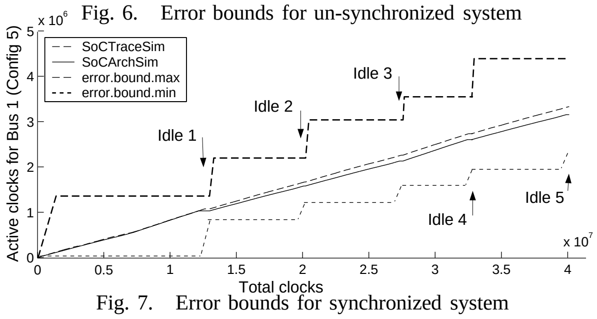

Fig. 7. Error bounds for synchronized system

and flexibility (multiple mappings, one-time trace generation) offered by SoCExplore cannot be ignored. Presently we are concentrating our efforts on fine-tuning techniques to establish more accurate bounds on trace-based simulation results which will ultimately add to the value of SoCExplore.

REFERENCES

[1] S. Abdi et al.,"System-on-Chip Environment - Tutorial", *CECS Technical report 02-28*, Sept 24, 2002.
[2] L. Benini and G. De Micheli, "Networks on chip: a new SOC paradigm", *IEEE Computer*, pp.70-78, 2002.
[3] A. Brinkmann et al, "On-chip interconnects for next generation system-on-chips", *15th Annual IEEE International ASIC/SOC Conference*, 2002.
[4] W. J. Dally and B. Towles, "Route packets not wires: on-chip interconnection networks", *DAC*, 2001.
[5] P. Kalla, X. Sharon Hu and J. Henkel, "A trace-based approach for on-chip network performance and energy analysis", TR-2004-27, Dept. of CSE, University of Notre Dame, 2004.
[6] F. Karim, A. Nyugen and S. Dey, "An interconnect architecture for networking systems on chips",*IEEE Micro*, 2002.
[7] S. Kim, C. Im and S. Ha, "Schedule-aware performance estimation pf communication architecture for efficient design space exploration", *CODES+ISSS 2003*.
[8] T. Kogel et al, "A modular simulation framework for architectural exploration of on-chip interconnection networks ", pg:7-12, *CODES+ISSS 2003*.
[9] K. Lahiri, A. Raghunathan and S. Dey, "System-level performance analysis for on-chip communication architectures", *TCAD*, 2001.
[10] http://cares.icsl.ucla.edu/MediaBench/
[11] A. D. Pimentel et al., "Exploring Embedded-Systems Architectures with Artemis", *IEEE Computer Magazine*, pp. 57-63, Nov. 2001.
[12] www.simplescalar.com
[13] H. S. Wang, X. Zhu, L. S. Peh and S. Malik, "Orion: A power-performance simulator for interconnection networks", *MICRO 35*, Nov. 2002.
[14] V. D. Zivkovic et al, "Fast and accurate multiprocessor architecture exploration with symbolic programs", *DATE 2003*.