Performance Evaluation for System-on-Chip Architectures using Trace-based Transaction Level Simulation

T. Wild, A. Herkersdorf, R. Ohlendorf
Technical University of Munich
Institute for Integrated Systems
Munich, Germany
http://www.lis.ei.tum.de

Abstract

The ever increasing complexity and heterogeneity of modern System-on-Chip (SoC) architectures make an early and systematic exploration of alternative solutions mandatory. Efficient performance evaluation methods are of highest importance for a broad search in the solution space. In this paper we present an approach that captures the SoC functionality for each architecture resource as sequences of trace primitives. These primitives are translated at simulation runtime into transactions and superposed on the system architecture. The method uses SystemC as modeling language, requires low modeling effort and yet provides accurate results within reasonable turnaround times. A concluding application example demonstrates the effectiveness of our approach.

1. Introduction

In the design of System-on-Chip (SoC) solutions the definition of suitable architectures is a vital issue. Typical SoCs may consist of a broad range of IP modules like embedded processors, accelerator blocks, interface modules, a memory subsystem and specifically designed HW modules. All architectural resources are connected via a communication infrastructure that provides system-internal connectivity for the exchange of data and synchronization. Choosing an appropriate system architecture and mapping the application's subtasks to the resources are important tasks in the process of system level design ([1]).

As the design space opens up a vast variety of solution alternatives, the specification of SoC architectures cannot base on intuitive decisions, which rely on a rough inspection and an intuitive estimation of design parameters, and their influence on the quality of the solution. In order to meet time-to-market as well as design goals concerning performance, area and power, thorough investigations have to be carried out. Therefore, a systematic exploration of design alternatives is necessary that makes up a strong basis for the final implementation steps, both for the hardware and software parts of the system.

Starting with modeling and simulating such a system on RTL level using a hardware description language is not feasible because of the modeling effort, the simulation times and the inability to capture the behavior of mixed HW/SW systems. Therefore, the abstraction level has to be raised. Recently, new modeling languages have been developed in order to support designers in the early design stages on system level. Among others like SpecC or System Verilog, mainly SystemC ([2],[3]) has gained attraction for design exploration. SystemC allows modeling of SoCs on a high abstraction level and gradual refinement for design and verification purposes.

The definition of the system architecture, i.e. the allocation of architectural resources and the mapping of tasks under given optimization criteria, is the major step in system level design. As exploration is an iterative process a great number of different potential solutions have to be evaluated regarding their compliance with the design requirements. This means that the complexity to generate suitable models for performance analysis as well as the effort for evaluation have to be strictly limited, in order to allow the comparison of as much alternatives as required. On the other hand it is necessary to capture enough information with high accuracy for making reasonable design decisions.

For performance analysis a model is necessary that captures both the function and the characteristics of the architecture resources adequately. The intrinsic properties of the architecture as well as the execution behavior of the application's subtasks including the memory architecture have to be taken into account. Moreover, the workload resulting from external stimuli has to be considered.

In this paper we present a transaction level approach to evaluate system performance for SoC architectures using SystemC as modeling language. Traces are used to capture the behaviors of the application on a high abstraction level and their interaction on the architecture. This provides high simulation efficiency combined with easy reconfigurability

3-9810801-0-6/DATE06 © 2006 EDAA

of the underlying model to different resource and mapping configurations.

The remainder of the paper is organized as follows. In the following chapter we give an introductory overview of modeling approaches on system level. Section 3 contains a description of our method and describes its implementation in SystemC. In Section 4, we demonstrate the usage of our simulation approach for a generic network processor architecture. Section 5 concludes the paper and gives some outlook for further improvements of our system simulator.

2. Related work

Performance estimation on system level is a topic of intensive research. Many approaches have been proposed that rely on different concepts. A Network Calculus based approach is described in [4] that uses performance networks for modeling the interplay of processes on the system architecture. Event streams covering the workload and resource streams describing service of resources interact in performance components that are connected to a network. The resulting transformations enable the derivation of performance data like resource load values or end-to-end latencies. SymTA/S ([5]) uses formal scheduling analysis techniques and symbolic simulation for performance and timing analysis. One problem of exact methods is their limited ability to capture real workload scenarios. Therefore, in many cases simulation-based methodologies are used.

Transaction level models (TLM) have gained wide acceptance in the system level design community ([6]). Decoupling function from communication and defining interfaces that provide specific functions that can be used to model abstract communication enables stepwise refinement of TLMs. Nevertheless, TLMs are applied on different abstraction levels and for very different purposes ([7]). The major abstraction level relevant for architecture exploration is the level of concurrent processes. However, in both variants, without and with timing information, the abstraction is too high to enable capturing the influence of the communication on system performance. In order to meet the goals of fast evaluation and high precision we concentrate in the following on models that are very abstract in respect to functionality and precise concerning architecture.

Ptolemy ([8]) is a design framework that targets at modeling, simulation and design of embedded systems with special consideration of different models of computation, however, with the main focus on specification and code generation. The POLIS system ([9]) supports the designer in modeling and verification of applications represented as CFSMs and guides towards implementation. The commercial tool VCC was based on ideas of POLIS and included the support of multiprocessor systems, however with restricted support of application domains. Metropolis ([9]) is a design environment for all phases of the design process from concept to implementation. It addresses also performance evaluation through simulation and formal methods using a metamodel that can represent different design aspects like function and architecture models as well as their mapping.

SystemQ ([11]) applies transaction level modeling in SystemC and uses queuing networks to cover the behavior of system-level platforms. Click ([12]) is an approach for specifying packet processing functionalities in a very efficient way, however, without providing means for the evaluation of their performance on specific system architectures. StepNP ([13]) is a network processor evaluation platform that utilizes Click as input specification. The performance simulation part of StepNP uses a SystemC TLM, however, includes full functional models that are executed on ISSs.

Trace driven simulation techniques have widely been used in the performance evaluation of computer systems in general ([14]) or in the area of multiprocessor systems ([15]). In [16] and [17] traces recorded from the functional level are mapped to the architecture and are used in the refinement process of the architecture, especially of the communication infrastructure. These approaches mainly rely on traces related to a given architecture and use them in the refinement of the system. In our performance evaluation approach we use traces in a more general way for the specification of the functionality including the mapping of its subtasks to the architectural resources and description of SoC workloads.

3. Trace-based system simulator

3.1 Concept

A modular approach is followed for the abstract SoC model that can be used to build network processor architectures consisting of a variable number of modules communicating via a shared SoC bus, as depicted in Figure 1. The modules may be embedded RISC processors, HW accelerators for specific processing tasks and memory blocks, either on-chip SRAM or memory controllers for off-chip RAM. A buffer manager responsible for storing and retrieving variable sized packets in memory and a queue manager that administers output queues are attached to the bus as well.

As the abstract SoC model, implemented as a SystemC TLM, is intended to be used for the investigation of our FlexPath NPU concept ([18]) it has to provide models for its pre- and post-processors as well the path dispatcher. Pre- and post-processor modules offload the RISC cores and are used in both the ingress and egress data path.

An application that runs on a network processor usually comprises the processing of different types of packets that receive specific treatment according to their protocol stack

and some packet individual properties. The basic principle of our performance evaluation approach is to abstract the involved functionalities by its processing latencies and to cover only the interaction of the associated sub-functions on the architecture, represented as inter-SoC-module transactions, without actually running the corresponding program code. This abstraction enables higher simulation speed than an annotated, fully-fledged functional model. Each sub-function is captured as a sequence of transactions, also referred to as trace in this paper. The binding decision for the sub-functions is considered by storing the corresponding trace in the respective architectural resource. A resource may contain several traces, one per each sub-function that is bound to it. The application is then simulated by forwarding packet references through the system and triggering the traces that are required for processing particular data packets in the respective SoC modules.

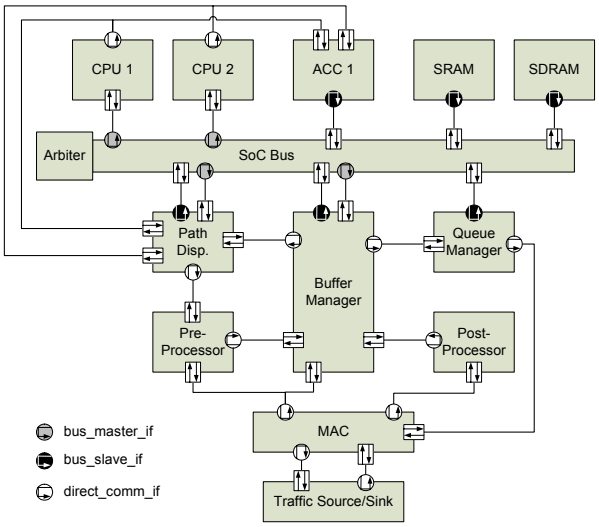

Figure 1: Architecture model

Each trace is made up of a sequence of tasks that are interleaved with transactions, representing the communication with other traces of different resources. Transfer of data to/from memory or external interfaces is handled in the same way by calling the traces of these components. For the sake of high simulation performance, processing is denoted simply by its particular execution latency on the respective resources. On the other hand, superimposing the transactions on shared resources and gathering usage data allows a precise evaluation of the architecture performance. The complete list of trace primitives used in our approach is given in Table 1, taking into account the different types of SystemC *sc_interface* that are currently used in our simulation environment.

Figure 2 shows a basic example for a CPU trace that performs some very basic packet processing. It begins with two read operations to fetch the packet reference and to read the required headers from memory. After a certain processing time - e.g. for checking the validity of the packet - a lookup is performed, represented by a write to a coprocessor and a read to return the result. Then packet processing is continued with an intermediate write operation and finally the modified parts of the packet are written back to the memory and the packet reference is sent e.g. to the queue manager.

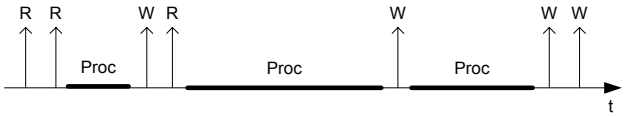

Figure 2: Example of a CPU trace

The transactions, in this case read and write, are annotated with the target module and the amount of data to be transferred and are issued on the CPU's bus interface. The bus model, in turn, maps the data volume into the corresponding number of words and – if the model covers this detail – number of burst transfers, performs arbitration and administers the usage of the bus. If the bus is available, the CPU requests are forwarded as a sequence of transactions to the interface of the particular slave, e.g. the SRAM or SDRAM block. In the respective memory, the traces for read or write are triggered considering also whether it is a single or burst access. In Figure 3 it is shown how a single read operation from SRAM and a burst read from SDRAM is transformed into traces specifying their response behaviors. The start of a transaction is indicated by an upward and the end by a downward arrow. Write operations are covered correspondingly.

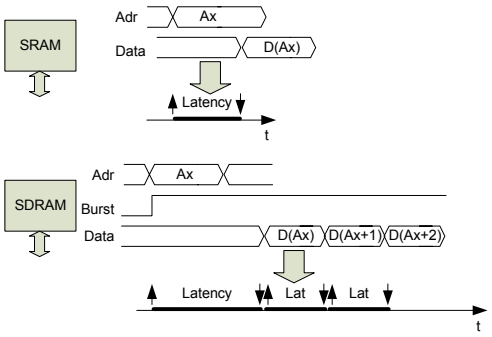

Figure 3: Abstraction of memory read operations

As mentioned above, performing transactions via a bus especially requires the consideration of additional delays caused by its character as a shared medium. Figure 4 shows this for the initial two read accesses in the CPU trace of Figure 2, assuming a single read from SRAM and a burst read of 3 words from SDRAM. The bus access times shown in the timing diagram are dynamically determined and contain latencies caused by both the arbitration process and the load situation of the bus.

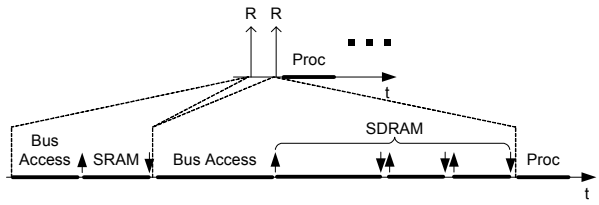

Figure 4: Expansion of a trace into a sequence of transactions

This short example shows how the influence of both the bus and properties of the involved architecture resources are considered during simulation. Interaction of architecture blocks via point-to-point links is also implemented as transaction, however, is handled straight forward without the overhead involved in bus based communication.

The trace specification for the system architecture is currently a manual process. The timing behaviors of the different resources can either be retrieved from data sheets, e.g. for memory or specific accelerator blocks, or by recording the activity of embedded CPUs with a logic analyzer in combination with a disassembler, that allows to sequence the program execution. A simpler starting point for trace specification are packet processing benchmarks like [19] that give typical instruction profiles for specific network processing tasks.

The complete processing of a packet in the system is thus defined by the sequence of traces. This sequence is predetermined by specifying an initial trace that is executed in the processor model and differentiates packets with particular protocol stacks. All further processing steps are then determined by this initial trace. Therefore, each incoming packet is annotated with the information about the entry point for the processing of the packet. Control dependencies as part of the processing are resolved in advance by assigning packets to different entry points. This procedure enables to consistently capture the processing of packets of the same type, i.e. of the same protocol stack (e.g. TCP over IP over Ethernet).

In order to capture the part of the behavior of real applications, which is directly related to specific properties of an individual packet (e.g. its size, the resulting priority and output port), a reference to a packet info containing these values, is forwarded through the simulator. At simulation time it is then possible to access this data and take it into account when translating the trace specification into the correct sequence of transactions. A good example for this issue is the packet size that is required in the buffer manager to store/retrieve the variable sized packets in/from memory. The external traffic workload is specified in a trace-like notation. It contains for each arriving packet the interarrival time from the preceding packet and all information that is needed on its path through the system architecture. In this way either artificial traffic or real network traffic downloadable in *pcap* format from the internet and transformed to the required trace format can be used for stimulation.

3.2 Implementation

The simulation model is modular and can easily be adapted to different architecture variants. On startup, a configuration file is loaded that specifies the properties of the SoC architecture to be simulated, especially the number and type of the resources as shown in Figure 1, and the data to be measured, e.g. load values of resources including the bus, queue fill levels or latencies of packets. Moreover, for each resource a file is loaded describing the traces, corresponding to the sub-functions that are mapped onto that resource.

Interaction between resources may occur mainly on two different parts of the communication architecture: either on the central SoC bus or on several point-to-point links - including interrupt lines - that connect the modules directly. For this purpose the following interfaces have been defined:

- *bus_master_if*: implements read / write on the bus
- *bus_slave_if*: implements arbitration (read_request / write_request) and read / write in the slave
- *direct_comm_if*: implements point-to-point read / write and interrupt on any resource type (where required)

For modeling the bus communication, a concept has been chosen that allows the usage of bus models on different levels of abstraction. In addition to an abstract, timed model, a cycle-accurate model has been developed that can be used for a more detailed investigation of bottlenecks in the system. For this purpose specific wrappers are available that adapt the high level models of bus masters and slaves to the detailed bus. Both bus models capture the behavior of a CoreConnect PLB bus including a priority based arbitration scheme, pipelined address phases, split transactions and concurrent transfers on the parallel read and write busses. Bus locks are not supported in both models, while timeouts are covered only in the cycle accurate version. Common transaction level models are located on a higher level of abstraction where these effects are frequently disregarded, leading to lower precision of simulation results.

Table 1 contains an overview of all trace primitives that are used to specify sub-functions in the different resource types of the simulator. Not all primitives are allowed in each resource type. This depends for example on the attachment to the communication architecture or the role as bus master or slave.

Each primitive is annotated with a certain number of parameters, like a latency value representing the duration of a packet processing subtask or a specific data amount for read and write operations (blocking or non-blocking). The variable size variants get the data amount via a packet

reference, as described before. Currently, a processing latency is a fixed value representing the required number of clock cycles. Variable processing latencies, e.g. for covering data dependencies, could easily be implemented in the same way as variable size data transfers. Function primitives that are translated to transactions with a different module of the architecture specify the target module and the trace number that has to be executed in the called module. Interrupts can be used in accelerators and on some of the modules in the ingress data path to cause CPUs to execute a specific trace as interrupt service routine, e.g. for reading back data or fetching packets for processing. The semaphore primitive is of special use to model data dependencies when performing parallel tasks. If a CPU has called an accelerator and continues its processing, a semaphore can be used to indicate a step in a trace where the result of the accelerator has to be read back at the latest, before processing can be continued.

Trace Primitive	Parameters
Bus Read specific size	Data amount, target, sub-funct
Bus Read variable size	Reference to packet info
Dcomm Read specific size	Data amount, sub-funct
Dcomm Read variable size	Reference to packet info
Bus Write specific size	Data amount, target, sub-funct
Bus Write variable size	Reference to packet info
Dcomm Write specific size	Data amount, sub-funct
Dcomm Write variable size	Reference to packet info
Processing	Latency
Issue Interrupt	CPU number, int. service routine
Semaphore	Slave number
End of trace	-

Table 1: List of trace primitives

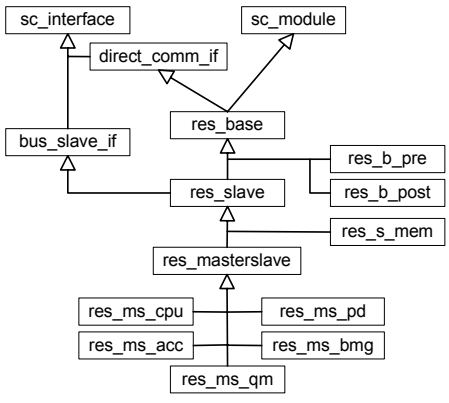

Figure 5: Class hierarchy of architecture modules

The different resource types as depicted in the architecture in Figure 1 are derived in a class hierarchy that adds increasingly more details specific to the respective type, as shown in Figure 5.

The implementations of the interface functions in the particular resource type allow controlling the interaction between resources and the bus, respectively, and triggering the execution of the required traces, realized as concurrent processes.

4. Experimental results

The trace based performance evaluation approach described above is now applied to investigate the architecture of a very simple network processor (NPU) with a single CPU for packet processing, as shown in Figure 6. The packets arriving from four independent MAC ports are segmented and stored in SDRAM by the buffer manager, the administration data for linking the segments belonging to a packet is held in SRAM. The CPU first fetches packet descriptors from the buffer manager and then the packet header from memory. After processing, data is written back and packet descriptors are sent to the queue manager, which determines when packets have to be retrieved from memory and sent out to the target network interface.

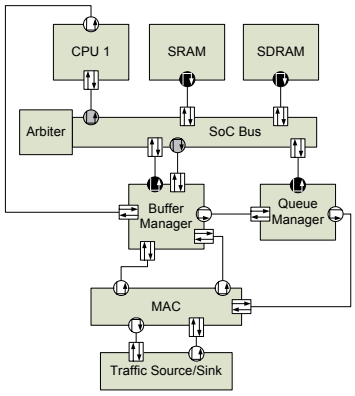

Figure 6: Simple NPU example

In this case a very basic processing scenario for IP forwarding ([19]) is modeled consuming 400 instructions per packet on the CPU, which needs 1.4 clock cycles per instruction. Input traffic consists of 64 byte packets. The following diagram shows the packet throughput (in million packets per second) of the system depending on the offered load with the processor clock speed as parameter.

The results of the evaluation in Figure 7 show that with increasing ingress load the NPU first gets into saturation due to overload of the CPU. Raising the clock rate of the CPU moves the saturation packet rate to a higher value, as expected. However, further increase of the load (beyond 1.5 Mpps) leads to a steep reduction of performance for all CPU speeds. An additional simulation with two CPUs running at 300 MHz shows a corresponding behavior. An analysis of further measurement data that are recorded during simulation shows that the increase of input traffic

leads to a bottleneck at the SDRAM interface. The buffer manger autonomously stores more packets than the CPU can process and thus reduces the memory bandwidth that is available for packet processing, throttling in turn the CPU. This example with initially unexpected results shows that it is very important to have a system architecture where the available resources are balanced and cooperate in an efficient manner. Exploration tools that help in finding such architectures are therefore of high importance.

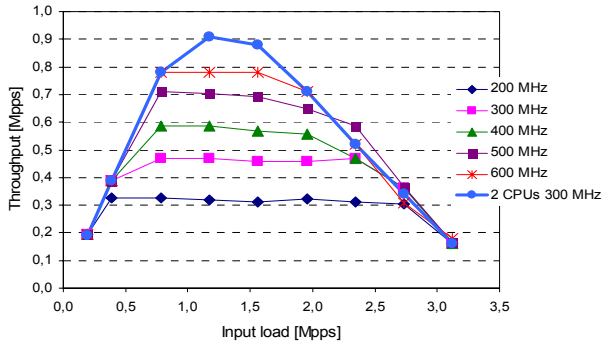

Figure 7: Simple NPU performance results

In order to adjust the model either concerning CPU speed or number of CPUs only a simple change of parameters in a configuration file was necessary. The runtime for all simulations contained in the diagram was about 10 min. on a 3.2 GHz Pentium 4 with 1 GB memory under Linux using SystemC 2.0.1. For generating the curves, a total of 270,000 packets have been simulated.

The experiment shows that our approach is very useful for investigating alternative architectural choices with very short turnaround times and for an interactive search in the solution space. In the experiment we have done this only for the processor complex. However, our simulation environment provides means to modify other parameters (e.g. number of resources, mapping of sub-functions to resources or queue sizes) in a similarly efficient way.

5. Conclusions and outlook

In this paper we have presented an efficient approach for performance evaluation as part of the architecture exploration process. The method is based on trace driven simulation using transaction level SystemC modeling technique with a very abstract specification of the functionality and yet very detailed coverage of the behavior of the architecture. This allows for a very fast and nevertheless precise investigation of SoC architectures.

Future work will be the support of more heterogeneous communication architectures with several buses or NoCs and the realization of a GUI that further eases configuration and measurement evaluation. The approach is generic in a way that it can also be applied in different application areas other than packet processing. One major extension will be the support of SoC architectures that are adaptable at run-time.

6. References

[1] K. Keutzer, et al., “System Level Design: Orthogonalization of Concerns and Platform-based Design”, IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol 19, 2000

[2] SystemC homepage, http://www.systemc.org

[3] T. Grötker, S. Liao, G. Martin, and S. Swan, “System Design with SystemC”, Kluwer Academic Publishers, Boston, May 2002

[4] L. Thiele, E. Wandeler, “Performance Analysis of Distributed Embedded Systems”, in R. Zurawski (Ed.), “Embedded Systems Handbook”, CRC Press, 2005

[5] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, R. Ernst, “System level performance analysis – the SymTA/S approach”, IEE Proc.-Comput. Tech., March 2005

[6] L. Cai, D. Gajski, “Transaction Level Modeling: An Overview”, CODES+ISSS 2003

[7] A. Donlin, “Transaction Level Modeling: Flows and Use Models”, CODES+ISSS 2004

[8] J. Buck et al., “Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems,” Int’l J. Computer Simulation, Apr. 1994

[9] F. Balarin et al., “Hardware-Software Co-Design of Embedded Systems: The Polis Approach”, Kluwer Academic Publishers, 1997

[10] F. Balarin, et al., “Metropolis, An integrated Electronic System Design Environment”, IEEE Computer, April 2003

[11] S. Sonntag, M. Gries, C. Sauer, “SystemQ: A Queuing-Based Approach to Architecture Performance Evaluation with SystemC”, Proc. Emb. Comp. Systems: Architectures, Modeling, and Simulation (SAMOS) 2005

[12] E. Kohler, et al., “The Click modular router”, ACM Trans. on Computer Systems, Vol. 18, No. 3, 2000

[13] P. Paulin, C. Pilkington, E. Bensoudane, “StepNP: A System-Level Exploration Platform for Network Processors”, IEEE Design&Test of Computers, Nov.-Dec. 2002

[14] S.W. Sherman, J.C. Browne, “Trace Driven Modeling: Review and Overview”, Proc. Symp. On Simulation of Computer Systems, 1973

[15] A. Prete, G. Prina, L. Ricciardi, “A Trace-Driven Simulator for Performance Evaluation of Cache-Based Multiprocessor Systems“, IEEE Trans. Parallel and Distributed Systems, Vol 6, No. 9, 1995

[16] P. Lieverse, P. van der Wolf, E. Deprettere, “A Trace Transformation Technique for Communication Refinement”, CODES 2001

[17] A. Lahiri, A. Raghunathan, S. Dey, “Fast performance Analysis of Bus-Based System-On-Chip Communication Architectures”, ICCAD 1999

[18] R. Ohlendorf, A. Herkersdorf, T. Wild, “FlexPath NP – A Network Processor Concept with Application-Driven Flexible Processing Paths”, CODES+ISSS 2005

[19] R. Ramaswamy., T. Wolf, “PacketBench: A Tool for Workload Characterization of Network Processing”, IEEE Workshop on Workload Characterization, October 2003