In: ANCHOR 2004, Advanced Networking and Communications Hardware Workshop, held in conjunction with the 31st Annual International Symposium on Computer Architecture (ISCA 2004), Munich - Germany, 19.-23. Juni 2004.

Developing an IP-DSLAM Benchmark for Network Processors

Gunnar Hagen[1], Jörg-Christian Niemann[2], Mario Porrmann[2],
Christian Sauer[1], Adrian Slowik[3], Michael Thies[3]
[1]*Infineon Technologies, Corporate Research, Munich, Germany*
[2]*Heinz Nixdorf Institute and Department of Electrical Engineering, University of Paderborn, Germany*
[3]*University of Paderborn, Germany*
{gunnar.hagen, christian.sauer}@infineon.com, {niemann, porrmann}@hni.upb.de,
{adrian, mthies}@upb.de

Abstract

DSL access multiplexers (DSLAMs) link the customer side of the network with the service side. In this domain, recent trends, such as end-to-end IP and QoS based protocols for better bandwidth utilization, require in-field reconfigurable, yet cost-effective solutions. This, however, suggests that common network processors cannot be used efficiently since they scale their resources linearly with the throughput class. The scaling limitations are due to the fact that they are either used stand-alone or within rather symmetric clusters, which makes them sub-optimal for the asymmetric topology of a DSLAM. Clearly, to develop an appropriate architecture a careful application domain analysis and a well defined benchmark are required. The availability of system-level network processing benchmarks, however, is still an unresolved issue. This is especially true for the network access domain, where no established benchmarks exist. In this paper, we present a complete system-level benchmark including function, traffic models, and environment specification. We first perform the domain analysis, derive its requirements, and examine existing approaches. We then apply the benchmark to a number of processors, including NP elements. Finally, we discuss our observations. This not only demonstrates the concept, but also shows how the benchmark can be used for the subsequent design space exploration step.

1. Introduction

Challenging applications and increasing line speeds continuously drive the development of network processor (NPU) architectures. These high-end devices are well suited for core network equipment or server equipment. However, high-volume equipment for network access, such as DSL access multiplexers (DSLAMs), requires solutions that put considerable effort into achieving a high performance cost-efficiency.

Necessary prerequisites for such cost-efficient solutions are system-level benchmarks and constraints that allow for comparing existing solutions and enable an application-driven design-space exploration for new architectures. Such system-level NPU benchmarks, especially for the access domain, are not available yet. Instead, existing network benchmarks focus solely on the micro-level and function-level. Therefore, we need to define a system-level benchmark based on a careful analysis of the application domain and its requirements. Based on our previous work [1], we believe that such a benchmark should not only include a functional specification but also requires a measurement specification and a system environment including suitable traffic models.

Here we focus on the domain of DSL access multiplexers. DSLAMs link the customer side of the network with the service provider side, as shown in Fig. 1. In this domain, recent trends, such as end-to-end Internet protocol (IP) and Quality-of-Service (QoS) based protocols require in-field re-configurability.

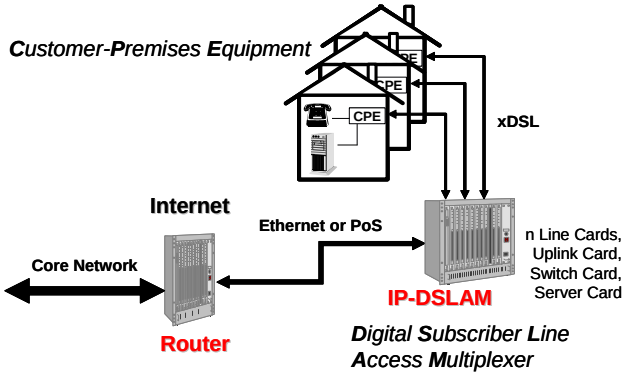

Fig. 1: IP-DSLAM deployment scenario

The figure highlights our future outlook on aggregation and transport technologies, where most of the equipment has become obsolete. Traditional narrowband switches as well as ATM switches have completely disappeared; former broadband remote access server (BRAS) functions have become partly obsolete, partly distributed. The IP-DSLAM represents now the central

access point for all narrow- and broadband services; the former parallel networks converged and left highly reliable routers as the dominating network element.

Besides the development of the IP-DSLAM benchmark, this paper focuses on its application to demonstrate the usefulness for the design space exploration. We therefore apply our benchmark to processors common to the domain, as well as 'standard' network processor elements.

The remainder of this paper is organized as follows. The next section analyzes trends in the application domain and elaborates the requirements of a benchmark and the DSLAM system. Section 3 discusses existing benchmarking approaches related to NPUs. In Section 4 we specify our system-level benchmark and describe its implementation. We apply the benchmark to a number of processors and discuss our observations with respect to design space exploration in Section 5. Finally, the paper is concluded in Section 6.

2. Application domain analysis and requirements

The IP-DSLAM presents a step in the evolution from a classical layer-2 restricted DSLAM towards a multi-service access platform of a converged network. It will integrate functions that were classically assigned to different types of network elements: Multiplexing (classical layer-2 DSLAM), trunk-switching (ATM cross connect), admission control and IP based routing (BRAS edge router).

Physically, the IP-DSLAM will still reside at the central office, supporting towards the customer side a high number of copper lines with various DSL-techniques (e.g. ADSL, VDSL and SHDSL), while supporting the regional network with only a significant smaller number of optical high-speed uplinks (of Gigabit Ethernet or Packet-over-Sonet type). Based on the number and type of user interfaces, bandwidth and flow requirements can be derived. Bandwidth requirements may thereby be asymmetric.

The idea in this context is to come up with a workload model that anticipates performance requirements for a point in time when a related network processor design could actually be manufactured. In this regard, some of the benchmark requirements depend on the correct anticipation of trends and take the risk of unexpected, disruptive changes.

To derive functional requirements, the categories of

- Application-trends,
- Protocol-trends,
- Network-trends, and
- Platform-trends,

will be reflected and projected towards the future.

Starting with a look on applications [2], the trend towards interactive voice- and video-services with their inherent real-time demands is obvious. Network gaming and conferencing add the requirement for multicast capabilities. The benchmark under discussion will reflect real-time requirements by a rule set that requires the classification of traffic into different service categories. Overall, we predict a rather always-on/always-connected type of Internet usage with permanently assigned IP addresses and their correct usage (relating to the IP source address) being enforced. For the long term we predict in this context a reduced role of tunnel protocols.

As trends in applications can be also uncovered by studying their distinctive traffic profiles in web traces, we can uncover from there that while webpage transfer and email file-transfer still constitute a dominant part of Internet traffic, the share of multimedia related traffic is continuously increasing. A distinction between personal and corporate traffic further highlights this trend. Thereby it may be concluded that a user-demand for multimedia real-time streaming services which leads to the deployment of multicast functions rather than to the replication of unicast channels, is pending. A blocking conflict of the past between multicast media distribution and digital rights management will be resolved by new techniques such as large scale distributed watermarking through encryption [3]. It may be also assumed in this context that other challenges such as incremental deployment and scalability issues might be overcome by what is referred to as overlay multicast control protocols [4]. To summarize this altogether, we expect a benchmark to support multicast operations with the additional feature of transforming a stateful multicast tree to a stateless (per-packet explicit) one.

Proceeding with a look on protocol trends we (still) see the ATM protocol slowly fading out. The functional decomposition of this complex protocol can only lead to two conclusions: At first, for handling simple data-link layer aspects Ethernet is about to become equally well suited as ATM. And as the vast majority of IP traffic is generated over Ethernet anyway, Ethernet is a natural candidate to replace ATM at the first/last mile as an in the end redundant protocol. Secondly**,** ATM will always be confined by its connection-oriented nature, while the homogenous combination of IP (connectionless) with MPLS (connection-oriented) has the potential to provide an economic solution even for short-lived traffic flows. In this context, we emphasize that a benchmark should reflect Quality-of-Service issues on the IP-layer (the end-to-end 'connectivity-layer'), in combination with IP related buffer-management mechanisms (e.g., a Token-Bucket scheme).

We further predict a variety of service-specific accountings, a principle enforcement of correct IP source address usage and an increased role of MPLS for traffic engineering purposes or setup of Virtual Private Networks.

When it comes to IP protocol versions we consider IPv6 relevant, however we estimate conservatively when it will actually start to dominate network traffic. The investments required to replace existing IPv4 infrastructure and the coordination effort to achieve a wide scale deployment beyond limitations on to newly created subnetworks will be challenging.

Going from here to network trends we assume that network operators will continue to be forced to get their operation expenditures down by consolidating their networks. Network consolidation however comprises the necessity to stepwise migrate functionality from one protocol to another. This results in inter-working requirements between the ATM-, Ethernet-, and IP protocol, but also potentially newly emerging protocols.

As a last step we will consider some aspects with regard to system platform trends. The IP-DSLAM [5][6], will include three types of modules, as shown in Fig. 2:

- Line-cards (Up to 32 with 96 ports max.)
- Uplink-card(s) (1 or 2)
- [Server-card(s) (1 or 2)]

For economical reasons (economy of scale factor) the switch-cards will be realized with cost-efficient Ethernet switches. The switch however will only support traffic aggregation towards the trunk cards and disaggregation towards the line-cards. We request Ethernet as a backplane protocol, which could be different from the layer-2 protocol applied at the subscriber-line (e.g., still ATM). Carried by Ethernet, we see NP-Forum's message layer protocol [7] utilized for a DSLAM internal message exchange, e.g., to support QoS aware flow control schemes, isochronous flows and unrestricted loop-bonding across chassis blades. This system area protocol will enable a flexible assignment of tasks among NPUs residing on the uplink-card and on the line-card. The role of server-cards is of no relevance for the formulation of the benchmark and will not be discussed.

Functional requirements of real DSLAM applications are not completely independent of the traffic direction (upstream, downstream) and the DSLAM locality (line-/uplink-card). Furthermore, they vary also – for the same type of equipment – with different configurations and deployment scenarios; a fact that characterizes multi-service equipment in general.

An NPU architecture will neither be optimal for all four of these isolated cases, nor would it be a realistic approach to deploy NPUs that are specialized for a specific traffic-direction, e.g., one for the upstream, another for the downstream traffic. This is why on some level a consolidation is inevitable.

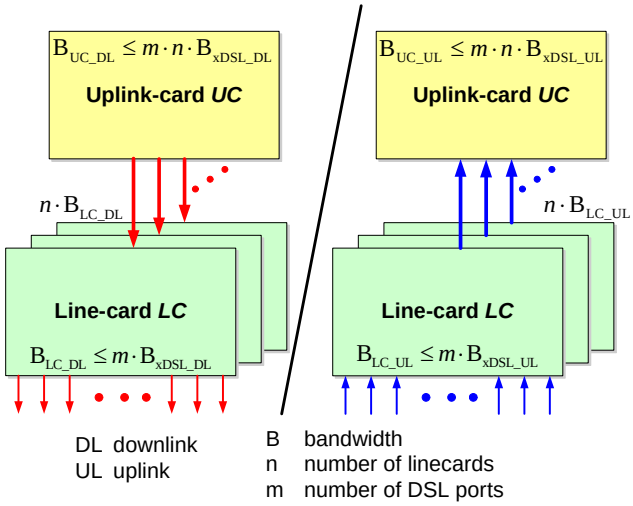

Fig. 2: DSLAM system components

With the approach chosen here, the consolidation is directly applied to the benchmark-level. This means that the synthetic nature of the benchmark mimics load requirements from the upstream and the downstream direction alike. However, given its modular nature, specific tasks can be excluded.

Finally, we believe that an IP-DSLAM benchmark should place special emphasis on the bandwidth characteristics of traffic flows (versus the packet processing needs). By definition, multi-service access platforms are characterized by the wide range of services that they need to support. Therefore, an IP-DSLAM benchmark must be able to uncover bottlenecks resulting, for instance, from insufficiently flexible system-on-chip communication or limited memory bandwidth.

The concept of the IP-DSLAM benchmark fulfills this due to the following facts:

- Spatial multicast (duplication of packets) significantly changes the bandwidth characteristic of a flow while it is being processed.
- Segmentation/fragmentation introduces overhead (header replications), which modifies the bandwidth characteristic of a flow.
- Segmentation/fragmentation requires access to the payload. This challenges architectures that relax chip internal bandwidth requirements by immediately separating header and payload to forward them on independent paths.

3. Existing approaches

After defining the requirements for our benchmark in the previous section, we examine and evaluate recent benchmarking approaches regarding their contribution to the benchmark. We are aware of currently eight primary benchmarking efforts related to NPUs: CommBench, EEMBC, MiBench, NetBench, NPF Benchmarking Working Group, LinleyBench, Intel Corporation, and NPBench. For most approaches, a discussion of their benchmarking methodologies is already provided in [1], we therefore focus in the following on a brief description of their functionality, the usefulness for the DSLAM scenario and the availability of specification and source code.

CommBench. CommBench [8] specifies eight task-level network processor benchmarks, four of which are classified as "header-processing applications" and the other four as "payload-processing applications". Implementation and analysis of CommBench target general-purpose uni-processors [8] and it is still unclear how their benchmark suite can be applied in the context of network processors. The CommBench source is implemented in C and available on request.

EEMBC. The Embedded Microprocessor Benchmark Consortium [9] defines a set of 34 benchmarks in areas of automotive/industrial, consumer, networking, office automation, and telecommunication domains. In the networking domain, EEMBC defines three simple task-level benchmarks (Patricia route lookup, Dijkstra's OSPF algorithm, and packet flow between queues). The benchmarks are implemented in C and include a precise measurement specification. A membership is required in order to obtain the source code.

MiBench. MiBench [10] follows a concept similar to EEMBC, but is composed from freely available source code. A set of 36 task-level benchmarks is defined in areas of automotive/industrial, consumer, office, networking, security, and telecommunication domains. In the networking domain, MiBench uses Patricia route lookup, Dijkstra shortest path search, and CRC32. The benchmarks are implemented in C and include a 'small' and a 'large' set of data for each kernel as work load.

NetBench. NetBench [11] defines and classifies a set of nine network processor benchmarks according to "micro-level" (e.g., CRC32, lookup), "IP-level" (e.g., IPv4 routing, NAT), and "application-level" (e.g., encryption, security) granularities. Results are presented for the SimpleScalar simulator and the Intel IXP1200. The C source code for the benchmark is freely available.

NPF Benchmarking Working Group. The NPF Benchmarking Working Group [12] (NPF-BWG) defines benchmarks based on 'micro-level', 'function-level', and 'system-level' applications. As of this writing, three function-level benchmarks are available: IP forwarding, MPLS, and a switch fabric benchmark. NPF-BWG addresses environment- and system-related NPU benchmarking issues, such as the workload and measurement issues in their specifications. However, no source code is available.

LinleyBench. The Linley Group benchmark [13] builds on the IPv4 definition of the NPF-BWG. In addition, DiffServ classification and marking, and an optional ATM-IPv4 interworking test are defined. The complete benchmark specification is commercially available. To date, results for the IBM network processor are published.

Intel Corporation. Intel's [14] benchmarking approach focuses on the IXP1200 network processor. They define a four-level benchmark hierarchy: hardware-level, micro-level, function-level and system-level. They provide results for IPv4 forwarding on the Intel IXP1200. Intel's executable description and method for performance measurement are specific to the IXP1200. The benchmark source code is not directly available, their NP framework, however, contains several sample applications, including IPv4 forwarding. Furthermore, Intel is contributing to the NPF-BWG.

NPBench. Lee and John [15] recognize the lack of control-plane benchmarks. They define a set of 26 functions (in three classes: traffic-management and Quality-of-Service, security and media processing, and packet processing), which target both the data and the control plane of network processors. Ten of these functions are implemented in C and compared to CommBench, for the rest they refer to [8][9][10]. The benchmark suite is available on request.

Other Work. Crowley et al. [16] present an evaluation of theoretical network processor architectures based on a programmable network interface model and detailed workload analysis on the application level. For the analysis, three of the identified loads, namely IPv4 forwarding and IP security (MD5, 3DES) are used. They do, however, neither provide sources nor detailed specifications of their benchmark.

Summary of Existing Approaches. In Table 1 we compare existing approaches to network processor benchmarking. As this table shows, most of the benchmarks are implemented in C. Such a high-level implementation in combination with a sufficient traffic model provides an executable specification, which helps to achieve functional correctness. The measurement specification, furthermore, allows to compare results provided by different vendors.

Table 1: Characteristics of existing approaches.

	Granularity	Source code	Availability	Traffic model	Measurement
Comm Bench	Micro	C	On request	No	No
EEMBC	Micro	C	Membership	Yes	Yes
MiBench	Micro	C	Free	Yes	No
NP Bench	Micro	C	On request	No	No
Net Bench	Micro/ Func	C	Free	Yes	No
NPF-BWG	Function	No	Spec only	Yes	Yes
Intel	Function	uE-C	No	N/A	IXP specific
Linley-Bench	Function	No	License	Yes	Yes

The granularity of the benchmarks can be distinguished in micro-level, function-level, and system-level, e.g., as used in [1][12][14]. Most suites define micro-level benchmarks, e.g., CRC32 and table lookup, which constitute the elementary tasks of functions. Some suites define function-level benchmarks. The most common function is IPv4 forwarding. None of the approaches, however, defines a system-level benchmark as needed for our IP-DSLAM scenario. In the next chapter, we therefore especially discuss the system-level specification, traffic model, and a couple of noteworthy implementation issues.

4. Benchmark specification and implementation overview

Reflecting DSLAM related internetworking trends we derive a couple of implications and requirements for the conception of the benchmark, and account for these within the implementation. In Fig. 3, the main components of our benchmark are shown, including the eight functional kernels, environment and traffic model.

A. Functional specification

We consider the following list of functional blocks to be representative for an IP-DSLAM benchmark.

1. **Parser**. Since we aim at software-defined functionality for packet processing, the first processing step needs to be the derivation and specification of the packet related data structure between frame delimiters of the received bit-stream. The required level of resolution for the parser is determined by the fact that subsequent tasks shall get all input data in an explicitly structured form. In our case this is up to the resolution of layer-4 information.
2. **Headercheck**. The correctness of data passed on to subsequent tasks must be verified. The verification of the IP packet header shall be done according to the requirements of RFC 1812 [17]. This includes the verification of the IP header checksum.
3. **Classification**. We further claim that the correct usage of the IP source address should be enforced (in upstream direction only) and that forwarding decisions should be made based on the IP destination address (in downstream direction only). The idea here is to base access control on a reverse-path forwarding mechanism. This means that we use the IP source address to determine the hypothetical destination port. Should the determined port be identical with the port the packet was actually received on, we declare the IP source address valid and do not drop the packet. We also require a 5-tupel micro-flow classification, based on the standard set of IP source/destination address, protocol type and layer-4 TCP/UDP port-numbers. In downstream direction the classification result can be used for traffic conditioning purposes, in upstream direction it can enforce service level agreement (SLA) derived traffic profiles. The IP address related lookups and classifications, however, represent approximately a comparable load, independently of the direction dependent input-fields or actual use of the results.
4. **MC Duplication**. We see the IP-DSLAM located at the endpoint of a multicast tree with the multicast support to be performed in two stages. Stage one (realized at the Uplink-Card) is the more complex one that relaxes the requirements for stage two (realized at the Line-Card). The IP-DSLAM benchmark will only be concerned with requirements of stage one. At this stage, multicast-support is not only provided by the simplified mapping of an IP-packet to an Ethernet broadcast or multicast address but by actual packet duplication, as long as packets are not bound for the same second stage. This is to prepare independent queuing for traffic shaping towards line-cards.
5. **NPF-ML**. Multicast packets that share the same second stage are not duplicated. Instead, an explicit list of identifiers is attached to the respective packets. Identifiers within this list refer to port addresses such that packets, once they arrive at the second stage, can be immediately forwarded to the respective ports without further consultation of group lists. The NP-Forum's message layer protocol (NPF-ML) [7] is used for that purpose. In summary, this mimics the load requirements of an explicit small group multicast implementation scheme for IP-DSLAMs.

6. **Policing & Conditioning**. For traffic shaping (downstream-direction), and policing (upstream-direction), we use a token-bucket scheme. The scheduling strategy is based on a simple priority scheduling mechanism. Although there are traffic-direction dependent tasks to be performed, the underlying algorithms and the resulting workload are considered to be comparable.
7. **AAL5**. Taking note of current internetworking scenarios, the benchmark shall include ATM's AAL5-segmentation layer.
8. **Ethernet**. We imply that we would use NP-Forum's message layer characterizing the payload-structure as transported by an Ethernet frame between the line-cards and the uplink-cards. In this context, we would only require that the number of ATM cells packed within the Ethernet frame is determined. This is very much in line with ATM-Forum's specification for frame-based transport of ATM over Ethernet, (cf. ATM FATE specification [18]). Finally, a simple Ethernet forwarding procedure takes over and transmits the frame.

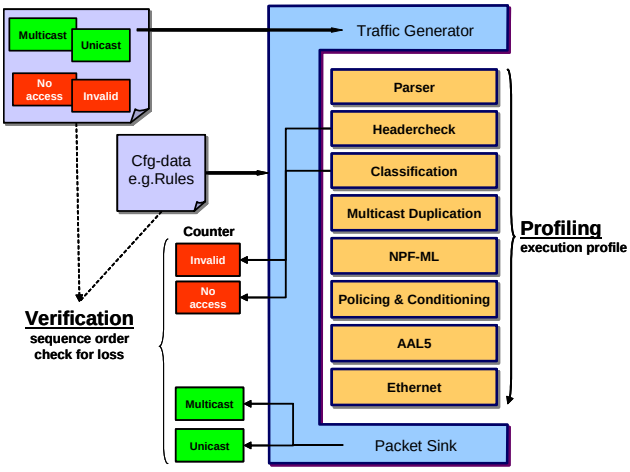

Fig. 3: Benchmark structure

B. Traffic model

The traffic model defines the workload for the previously defined benchmark. The model needs to fulfill three categories: address distributions, Quality-of-Service distributions, and packet size distributions.

Beginning with the address distributions we need to look at the layer-3 and layer-4 destination/source addresses and the port-numbers. As protocol types we declare in the same ratio UDP and TCP used. For the destination addresses we select from 1% to 9% in the range of multicast addresses; complementary we select 99% down to 91% in the range of unicast addresses. For multicast-traffic we foresee up to 32 multicast groups, each with up to 96 group members. Conservatively, we assume only one IP address per port and overall, a relatively contiguous address range for the IP addresses. IP addresses, however, will be randomly associated with line-card ports to mimic uncompressed search trees. We further imagine that the IP-DSLAM will support 96 ports per line-card along with 32 line-cards per System, what totals to 3072 different IP addresses. We select the IP source addresses from the same range as the IP destination addresses, restricted however to the unicast region. We assume up to 16 different flows per DSL-port that are only distinguished by their layer-4 destination port addresses.

The QoS distribution depends on both address distributions and the classification rules. We chose the rules in a way so that the classification results deliver about 50% expedited forwarding and 36% in assured forwarding (with its four subclasses along with three different drop probabilities each). The remaining 14% are declared best effort. This is a similar distribution as suggested with the LinleyBench. For the packet size distribution, we still follow a standard type of IMIX with its 7:4:1 packet-distribution, i.e., 7 packets with the size of 40 Bytes, 4 packets with the size of 552 Bytes and 1 packet of 1500 Byte. The sequence of packets within the benchmark file stream is determined in three steps:

- In the first step, we distribute packets randomly according to their IP destination addresses.
- In the second step, we group IP packets within certain sections according to their size. The intention here is to create bursts of minimum packet sizes that could potentially be observed for packet loss without influencing the overall IMIX profile. In this regard, the traffic-mix is intentionally not self-similar.
- In the third step we insert bursts of packets that preset stress situations for the header-verification task. A compliant execution of the benchmark must be able to maintain management information base counters even during these short periods. However, the overall load created by non-compliant packets shall be minimized.

To support verification issues, we finally copy sequence numbers to the payload of each IP packet (without changing the packet size).

C. Measurement specification

To allow for a simple comparison of different NPU cores, we suggest to normalize the benchmark results as follows:

$$\frac{\sum_{t \in tasks} cycles_t}{8 \cdot \sum_{p \in packets} size_p [Byte]} = Performance \left[\frac{cycles}{bit} \right] \quad (1)$$

The term "tasks" relates in this context to the eight functional kernels of the benchmark. Since the benchmark implementation conforms to ANSI C, it is portable with little effort to other target architectures. Obviously, a cycle accurate simulator is a prerequisite to the evaluation of the performance given above in (1).

D. Implementation

The implementation of the IP-DSLAM benchmark conforms to the current ANSI-C standard. It therefore neither expresses explicit parallelism nor does it contain any mappings of source code fragments to processing elements. The source code is organized into 37 files with a total of 4520 lines of code. All program objects are allocated within a single, global address space. There is no explicit memory hierarchy to provide non-uniform memory access.

The benchmark consists of a main loop that drives the entire packet processing. This main loop terminates if either a predefined volume of packet data or a predefined number of packets have been processed. At the very beginning of every iteration, the main loop calls the packet generator and fetches the packet descriptor of the next packet to be processed. The main loop then repeatedly calls the functions that implement the main IP-DSLAM functional tasks we have described. Some of these tasks may indicate that the current packet has to be dropped for reasons of access control. In this case the main loop increments appropriate event counters and proceeds with the next iteration. Otherwise, it passes the packet descriptor on to the next function that has to be executed. Multicast packets may lead to iterations of a nested inner loop that replicates the packet for every line-card taking part in the multicast distribution tree. This inner loop obviously only calls those functions that relate to downstream functionality.

The workload of the DSLAM is encapsulated within the packet generator. At initialization time, it precomputes a set of destination addresses, application ports, and other characteristic attributes. Particular settings that affect the configuration of the DSLAM and hence also affect the packet generator have to be fixed at compilation time. All parameters that contribute to the configuration of the DSLAM, like line-rates, number of ports and line-cards, are located within a single header file for ease of configuration.

5. Results

In this section we present first results of the proposed benchmark and show how the benchmark can be used to find a system architecture that matches the requirements of a given application scenario. We start our analysis with two embedded processors. One is a general purpose processor while the other one has an instruction set that is optimized for protocol processing. If a single processor solution does not fit the requirements of a given benchmark scenario we will show how the performance can be increased by adding application specific instructions, co-processors or by increasing the number of processors.

The main components of the next generation DSLAM that we will focus on in the following are an uplink-card, which controls and aggregates the traffic to the Internet Service Provider (ISP), and up to n=64 line-cards, which make in each case up to m=96 xDSL ports available (cf. Fig. 2). In order to evaluate the maximum computational requirements for such a system with the traffic model that has been defined in the last section, a worst case scenario concerning the bandwidth requirements is applied. For example, the aggregated bandwidth B_{LC_UL} of the line-card upstream is given by the sum of the maximum input bandwidths B_{xDSL_UL}. The input bandwidth B_{LC_DL} of the line-card downstream is given by the sum of the maximum output bandwidths B_{xDSL_DL}. The aggregated bandwidth of the uplink-card B_{UC} in either direction linearly scales with the bandwidth provided by the n line-cards cf. (2).

$$B_{UC\_UL} = m \cdot n \cdot B_{xDSL\_UL} \text{ , with } B_{xDSL\_UL} = \begin{cases} 0.8\,\text{Mbps for ADSL} \\ 3\,\text{Mbps for VDSL} \\ 2\,\text{Mbps for SHDSL} \end{cases}$$

$$B_{UC\_DL} = m \cdot n \cdot B_{xDSL\_DL} \text{ , with } B_{xDSL\_DL} = \begin{cases} 8\,\text{Mbps for ADSL} \\ 22\,\text{Mbps for VDSL} \\ 2\,\text{Mbps for SHDSL} \end{cases} \quad (2)$$

E. Target architectures

The first target architecture for our performance analysis is a 32 bit general purpose RISC processor, called N-Core [19][20]. The N-Core is a two address machine with a straightforward load/store architecture. Two banks of sixteen 32-bit registers can be alternatively used. Each instruction has a fixed length of 16 bits resulting in a high code density. The execution of most instructions takes one clock cycle. The N-Core features a short three-stage pipeline and an addressing interface that supports byte granular access. Since the processor is a soft core, it can be easily extended with specialized hardware.

The second processor, NPU-Core, is an application-specific 32bit RISC core that is tailored to packet process-

ing tasks, similar to the one described in [21][22]. The architecture provides instructions with subword bit-level access and support for multiple threads in hardware. As seen before, most instructions can be executed in one cycle. The deeper pipeline, however, allows for nearly two times the clock frequency when compared to the N-Core.

F. Exemplary benchmark results

As described in the last chapter, the different tasks have been implemented in ANSI C. The benchmark implementation has been profiled on both of the presented processor cores with the underlying modified IMIX workload model that has been introduced in section 4. The generated workload comprises 2981 packets with a total size of 0.955 MByte. The benchmark results are summarized in Fig. 4. The cycle counts of the NPU core shown in the figure are normalized to account for the difference in clock frequencies.

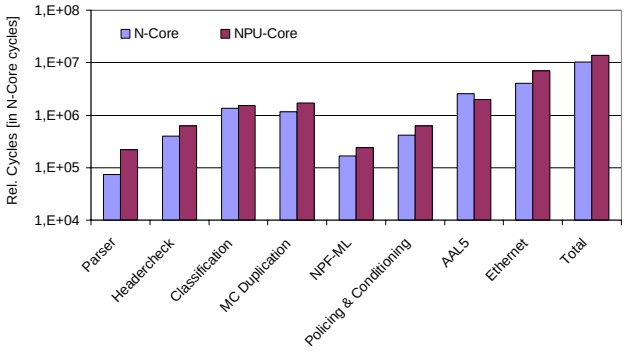

Fig. 4: Computational requirements of benchmark tasks for the embedded processors

As can be seen, the most computationally intensive tasks are AAL5 segmentation and Ethernet Encapsulation/Forwarding, which in both cases is due to CRC computations and explicit move operations applied to the entire packet. Surprisingly, with the exception of AAL-5, the NPU-Core performs worse than the general purpose core. On average, 3.5 cycles are required per bit of the packet stream when using the NPU-Core compared to less than 1.3 cycles per bit on the N-Core. Although this discrepancy is attenuated by the higher clock frequency of the packet processor, the general purpose processor still outperforms the ASIP.

A thorough evaluation of the different tasks and of their mapping onto the two processors is performed in order to reveal the performance bottlenecks of the architectures. The instruction set of the packet processing unit is widely reduced to special purpose functionalities for packet processing, thus providing only a basic set of general purpose instructions. Our analysis shows that the performance improvements that are achieved with the specialized instructions are more than annulled by the missing general purpose instructions. The most important reason is that the special purpose instructions, like bit field extraction, account for 10% of the executed code only, while 90% are general purpose instructions. General purpose instructions are mostly used to manage function calls and to access variables in stack frames. Moreover, the general purpose instructions of the packet processing unit are not suited for efficient compilation of common ANSI C constructs. Improvements could be achieved through fundamental restructuring of the compiler, or by handcoding critical parts in assembler.

As a case study, we have defined an IP-DSLAM 'Light' scenario, which reflects the asymmetric structure of the application. Therefore, only a subset of the functionalities is composed for the different system components and their corresponding transmission directions. This approach is chosen to demonstrate the flexibility of our benchmark, which can be easily adapted to different DSLAM system implementations. Table 2 gives an impression of the number of clock cycles that are required for the various benchmark tasks depending on the considered DSLAM component for the N-Core, using the same parameters as mentioned before. Only the tasks that are relevant for the given DSLAM component and traffic direction are considered, reflecting the typical asymmetric topology of a DSLAM. The N-Core processor requires 1.167 cycles/bit for upstream and 1.255 cycles/bit for downstream on the line-card; while 0.061 cycles/bit are needed for upstream and 1.986 cycles/bit for downstream on the uplink-card, respectively.

Table 2: Computational requirements of benchmark tasks for the DSLAM components

N-Core	Line-card		Uplink-card	
	Downstream	Upstream	Downstream	Upstream
Parser	74500	74500	74500	74500
Headercheck	396373	399353	396373	417233
Classification	1358874	-	1358874	-
MC Duplication	1156720	-	1171597	-
NPF-ML	-	-	169571	-
Policing & Conditioning	414587	-	414587	-
AAL5	2558393	2543493	-	-
CRC Accelerator	4093931	6024563	99099	-
Ethernet Framing	-	309920	349886	-
CRC in Software	43023639	63901032	966966	-
Total	**10053378**	**9351829**	**4034487**	**491733**
Cycles/bit	1.2550	1.1674	1.9855	0.0614

In order to give an impression of the performance requirements for different fields of application that can be evaluated with the benchmark, we have estimated the number of processors that have to work on the tasks in parallel under the assumption that a constant parallelization overhead of 10% is required.

Table 3 gives the number of N-Cores and NPU-Cores that are required in the proposed worst case scenario, as-

suming 96 ports per line-card and a CPU clock frequency of 300 MHz for the N-Core and 600 MHz for the NPU-Core, respectively.

Table 3: Required processing units for a line-card in the given DSLAM scenario

DSL version	Required N-Cores		Required NPU-Cores	
	Uplink	Downlink	Uplink	Downlink
ADSL	1	4	1	5
VDSL	2	10	2	14
SHDSL	1	1	2	2

Obviously, the highest performance is required for VDSL downlink. An on-chip multiprocessor with 10 N-Cores or 14 NPU-Cores and an appropriate communication infrastructure will be able to handle the required load even under worst case conditions. Exploiting parallelism, as described above, is one way to enhance the performance of the system, but it is expensively achieved. Here, it is mainly used to illustrate and compare the suitability of different processors for the execution of the proposed benchmark.

G. Design space exploration

Additionally, the benchmark is well suited for design space exploration as illustrated in the following. For the hardware implementation of a network component, we start with a single processor core. If the benchmarking shows that the single processor performance does not satisfy the design constraints, we will first try to optimize and modify the processing unit in order to speed up the execution of the target application. Only if this approach does not fulfill the requirements, parallel processor cores will be implemented.

Thus, the first step is to implement instruction set extensions, which (for the customer in a transparent way) are exploited by the compiler [23]. Therefore, frequently used instruction pairs are extracted and modeled as more efficient “super instructions”. Instruction pairs do not simply consist of adjacent instructions, but exploit direct data flow dependencies, (i.e., they share data in registers).

The next step for customizing the hardware is to attach more complex hardware accelerators or coprocessors to the CPU, which facilitate an even higher speed up. The higher performance comes at the cost of additional chip area. While appropriate “super instructions” can be identified and implemented with moderate effort, it is much more time consuming to identify larger tasks that can be efficiently implemented in application-specific hardware. Nevertheless, application-specific hardware has a much higher impact on performance reduces code size and lowers energy consumption [17].

As a last resort, we revert to coarse grained parallelism at the core level. Applying our benchmark to the expanded processing element, allows us to estimate the number of processing elements that are necessary to handle the given workload.

6. Conclusions

In this paper, we have specified and applied a benchmark for IP-DSLAM applications that reflects current trends within access networks. We applied the benchmark to two processors, a standard RISC-core and a common network processor element, and compared the results. Currently, we are actively extending the scope of our benchmarking to incorporate standard processors common to the domain, such as MIPS, PowerPC, and ARC.

So far, the main conclusions of our work are:

- We believe that a benchmark should anticipate requirements of current trends on the system-level in order to allow for sufficient lead time.
- A system-level benchmark can incorporate task-level benchmarks with well defined interfaces; for a realistic result, however, the execution of the complete benchmark is mandatory.
- Design space exploration based on our benchmark forms an effective approach to characterize the performance for IP-DSLAM elements to be developed.
- Specialized NPU processor elements depend on availability and quality of a compiler. Therefore, the standard processors appear to be better suited for our C level benchmark. A performance optimized implementation in assembly, however, could change the picture.

Acknowledgements

The research described in this paper was funded by the Federal Ministry of Education and Research (Bundesministerium für Bildung und Forschung), registered there under 01M3062A. The authors of this publication are fully responsible for its contents. This work was supported in part by Infineon Technologies AG, especially the department CPR ST. This work is part of the efforts of the GigaNetIC research project. The authors would like to thank Ulrich Ramacher, Ulrich Rückert, Uwe Kastens, and all other GigaNetIC members for their invaluable feedback.

References

[1] M. Tsai, C. Kulkarni, C. Sauer, N. Shah, K. Keutzer, "A Benchmarking Methodology for Network Processors," In Network Processors Design: Issues and Practices. P. Crowley, M. Franklin, H. Hadmioglu, P. Onufryk, 7, 1, Morgan Kaufmann Publishers, 2002.

[2] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, F. True , "Deriving Traffic Demands for Operational IP Networks: Methodology and Experience," In IEEE/ACM Trans. on Networking, June 2001

[3] I. Brown, C. Perkins, J. Crowcroft, "Watercasting: Distributed Watermarking of Multicast Media", In 1st Int. Workshop on Networked Group Communication, Pisa, November 1999.

[4] J. Liebeherr, T. K. Beam, "Hypercast: A Protocol for Maintaining Multicast Group Members in a Logical Hypercube Topology," In Proc. of 1st Int. Workshop on Networked Group Communication", July 1999

[5] Extreme Networks, "Intelligent Network Access," Whitepaper, 2004, http: //www.extremenetworks.com

[6] Extreme Networks, "Alpine 3800 Series," Data Sheet, 2004, http://www.extremenetworks.com

[7] E. Johnson, M.Lerer, "Message Layer Implementation Agreement," Network Processing Forum, January 2004, http://www.npforum.org/techinfo/approved.shtml

[8] T. Wolf, M. Franklin, "CommBench — A Telecommunications Benchmark for Network Processors," IEEE Int. Symp. on Performance Analysis of Systems and Software, Austin, TX, Apr. 2000, pp. 154-162.

[9] Embedded Microprocessor Benchmark Consortium (EEMBC), http://www.eembc.org/.

[10] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," In 4th Annual Workshop on Workload Characterization, Austin, TX, December 2001.

[11] G. Memik, B. Mangione-Smith, W. Hu, "NetBench: A Benchmarking Suite for Network Processors," Int. Conference on Computer-Aided Design (ICCAD), November 2001.

[12] S. Audenaert, P. Chandra, "Network Processors Benchmark Framework," NPF Benchmarking Workgroup, http://www.npforum.org/.

[13] LinleyBench 2002, http://www.linleygroup.com/benchmark.

[14] P. Chandra, F. Hady, R. Yavatkar, T. Bock, M. Cabot, P. Mathew, "Benchmarking Network Processors," In Network Processors Design: Issues and Practices. P. Crowley, M. Franklin, H. Hadmioglu, P. Onufryk, 2, 1, Morgan Kaufmann Publishers, 2002.

[15] B. K. Lee, L. John, "NpBench: A Benchmark Suite for Control Plane and Data Plane Applications for Network Processors," In Proc. of the Int. Conference on Computer Design (ICCD'03), San Jose, Oct, 2003.

[16] P. Crowley, M. Fiuczynski, J. Baer, B. Bershad, "Characterizing Processor Architectures for Programmable Network Interfaces," In Proc. of the Int. Conf. on Supercomputing, Sante Fe, N.M., May 2000.

[17] F. Baker, "Requirements for IP Version 4 Routers," Request for comments (RFC) – 1812, IETF Network Working Group, June 1995.

[18] "Frame-based ATM transport over Ethernet," Specification FBATM-0139.001, ATM Forum, July 2002, http://www.atmforum.com/standards/approved.html,

[19] J.-C. Niemann et al, "A holistic methodology for network processor design," In Proc. of the Workshop on High-Speed Local Networks, 28th Conference on Local Computer Networks (LCN2003), 583-592, 2003.

[20] D. Langen, J.-C. Niemann, M. Porrmann, H. Kalte, U. Rückert, "Implementation of a RISC Processor Core for SoC Designs – FPGA Prototype vs. ASIC Implementation," In Proc. of the IEEE Workshop on Heterogeneous reconfigurable Systems-on-Chip, 2002.

[21] X. Nie, L. Gazsi, F. Engel, G. Fettweis, "A New Network Processor Architecture for high-speed Communications," IEEE Workshop on Signal Processing (SiPS), 1999.

[22] J. Wagner, R. Leupers, "C Compiler Design for a Network Processor," IEEE Transactions on CAD, November 2001.

[23] U. Kastens, D. K. Le, A. Slowik, M Thies, "Feedback Driven Instruction-Set Extension," In Proc. of ACM SIGPLAN/SIGBED 2004 Conf. on Languages, Compilers, and Tools for Embedded Systems (LCTES'04), Washington, D.C., USA, June 2004.