A Comparison of Parallel Programming Models of Network Processors

Carsten Albrecht, Rainer Hagenau and Erik Maehle
Institute of Computer Engineering
University of Lübeck
{albrecht,hagenau,maehle}@iti.uni-luebeck.de

Andreas Döring and Andreas Herkersdorf*
IBM Zurich Research Laboratory
ado@zurich.ibm.com
A.Herkersdorf@ei.tum.de

Abstract: Today's network processor utilize parallel processing in order to cope with the traffic growth and wire-speed of current and future network technologies. In this paper, we study two important parallel programming models for network processors: run to completion and pipelining. In particular, the packet flow of a standard network application, IPv4 Forwarding, through two examined network processors, IBM PowerNP NP4GS3 and Intel IXP1200, is reviewed and characterized in respect to their programming models. Based on a benchmark for PC-cluster SANs, their application throughput and latency for Gigabit Ethernet is investigated and compared to a commercial, ASIC-based switch. It is shown that in this scenario network processors can compete with hard-wired solutions.

1 Introduction

With the tremendous growth of the Internet, high-performance network components, especially routers and switches, are under pressure to keep up with the traffic volume and speed rates of fiber-optic transmission lines. Today, they are mainly based on ASIC (Application-Specific Integrated Circuits) technology in order to meet the high requirements of fast packet processing. However, ASICs have a long development cycle of a year or more and often cannot be adapted to protocol or standards changes at the rate these occur in the dynamic world of networking. So, a new type of special-purpose processors optimized for fast packet processing in the data path of routers and switches has attracted much attention recently. These *Network Processors (NPs)* are not only able to analyze packet headers for packet forwarding and classification with wire-speed performance, but also shall take over more complicated tasks such as encryption/decryption, packet filtering for firewalls, or virus checking (deep packet processing).
However, the high performance required for wire-speed processing is only possible with parallel processor architectures tailored specially to this task. As these are completely new devices, standard solutions or compatibility requirements to earlier generations do

*now at Institute for Integrated Systems, Munich University of Technology

not yet exist. As a consequence, a rich variety of designs are seen which are difficult to judge in terms of their performance and potential. Therefore, in this paper a comparative study of two popular NP architectures is described, the IBM PowerNP NP4GS3[1] [Ib00] and the Intel IXP1200[1] [In01a]. The main focus of this study is on parallel programming models and performance. Both architectures essentially differ in the programming models propagated by the specific vendor. In general, the PowerNP is based on run to completion, processing each packet by a single thread. IXP applies pipelining, processing a packet by multiple threads.
In this paper we will first give an overview of the basic programming models for NPs in Section 2. In Section 3 the packet flow of both architectures is shown, and processing is compared with respect to a single packet. Then, an experimental setup based on cluster computing with Gigabit Ethernet utilizing an application-level benchmark and its results are presented in Section 4. Section 5 contains our concluding remarks.

2 NP Programming Models

The design and the development of NPs yield different approaches to fulfilling required qualities. On the basis of parallel processing one can distinguish two basic programming models for NP architectures: run to completion and pipelining.
The first model, called *run to completion*, is derived from concurrent processing. Incoming PDUs (protocol data unit) as data to be processed are read out from an ingress buffer. A number of processing units (PUs) build a pool of equitable processing engines. All PUs perform their operations simultaneously on their specific PDU fetched from the ingress buffer. After completing its task the PDU is written into an egress buffer, and the PU starts execution on a newly fetched PDU or is idle, i.e. waiting for work. Every PU runs the same application code. A PDU is completely processed by a single process or thread. Here, the application code is quite complex and needs much instruction store because it includes every functionality demanded by data-plane packet processing.
A characterization of this model reveals various advantages and disadvantages. It is a good starting point for a parallel network processing implementation because it allows a simple proceeding in the software-development process. One application code for all PUs means comfortable programming. Furthermore, the software can be implemented independently of the processor count. So, performance scaling by additional processors is only limited by hardware. Integration of new functionalities has no impact on the system model. However, it entails the risk of load imbalance overhead, which has to be avoided by an efficient load-distribution mechanism. High performance execution of dedicated tasks needs coprocessor support. This is difficult to integrate with a large number of concurrent PUs that would compete to access a common coprocessor. To reduce such coprocessor contention, replicated sets of coprocessor servicing a group of processors have to be integrated. For a moderate number of processors, the coprocessor queues suffice to handle this competition. A more important aspect is thread scheduling. After coprocessor invocation a thread becomes idle. The PU therefor needs additional threads to utilize its computational power efficiently (multithreading).

[1] In this paper, we use the abbreviated term *PowerNP* for the IBM PowerNP NP4GS3 and *IXP* for the Intel IXP1200

In contrast, the second model is derived from pipelined processing and is called *functional pipelining model*. Here, all PUs establish their own stage of a pipeline. An incoming PDU is read out from the ingress buffer, and the first process or thread executes its specific task on this PDU. After completing its work, it passes the PDU to the next process/thread in the pipeline. When the last PU has finished its operations, the PDU is completely processed and is written into the egress buffer for forwarding. All processes/threads run different application codes per stage. This application code is more compact than the software needed for the concurrent execution model because the implemented task is specialized and thus consumes less code. Each PDU is processed by all PUs of the pipeline.
A profitable aspect of the functional pipelining model is its focus on simpler subtasks. Thus, less functionality per processor enables lower hardware effort for instruction access per processor, e.g. a smaller instruction store. Specialization of a processor is also possible. So, this model provides some interesting freedom, useful for optimizing the hardware design. On the other hand, the functional pipelining model requires a larger effort in software implementation. The software has to be partitioned into subtasks of equal length. This could lead to idle times of PUs in some pipeline stages because code cannot always be partitioned ideally. Furthermore, in some situations the effort to process a certain task can generate varying execution times, e.g. payload processing tasks. The functional pipelining model reacts quite sensitively to such dynamic runtime length differences. Basic inter-stage communication and synchronization are needed for pipeline operation. Such interfaces must be built in hardware as well as software, which is tightly bound to the hardware design chosen.

3 Packet Flow Analysis

To compare the two NP platforms, PowerNP and IXP, with respect to their underlying programming models, a typical application is examined. As no open-source applications exist for NPs, we decided to use reference code provided by the board packages. The selected application has to fulfill the same level of service in networking on both platforms. Thus, we use IPv4 packet forwarding for packet-flow analysis that is available for PowerNP and IXP.

3.1 IBM PowerNP NP4GS3

The PowerNP's multiprocessor architecture, shown in Figure 1, consists of 16 picoprocessors (PEs) with numerous coprocessors. The PEs are organized in Dyadic Protocol Processing Units (DPPUs). Each DPPU contains two PEs, a small shared memory (scratchpad) and coprocessors such as a two-fold interfaced Tree Search Engine (TSE0, TSE1) for off-processor fast table look-up, and additional ones for tasks such as Data Store (DS), Checksum (CHK), Enqueue (ENQ), Counter (CTR), Semaphore (Sem), etc. [Al03, Ib02a, Kr01]. These hardware functions reduce lines of code processed by PEs because they provide access to and management of complex data structures. To increase its efficiency each PE is two-fold multithreaded so that each DPPU processes four packets concurrently [Ib02a].
The packet flow of a single packet through the EPC starts with dequeuing the packet by

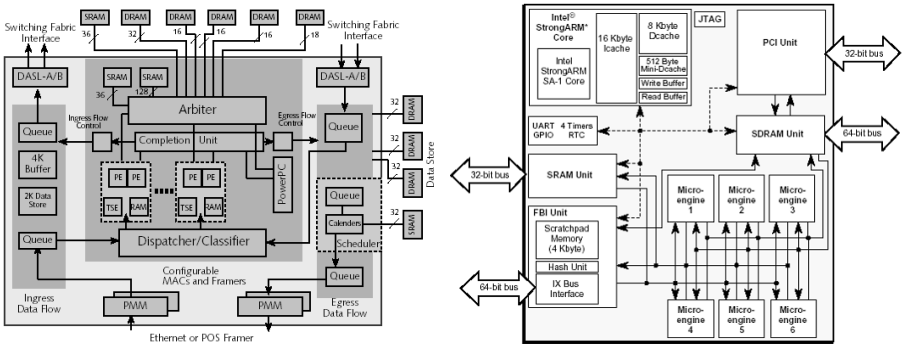

Figure 1: Block diagram of IBM PowerNP NP4GS3 [Kr01] (left) and Intel IXP1200 [In01c] (right).

the Dispatcher that also fetches initial data and assigns them to idle threads. Ingress and egress queues are selected round robin. Then, the Hardware Classifier decides how to process the incoming packet further. Extracting and evaluating several header options, it parameterizes the thread invoked with a code entry point, preloaded register contents and header data in scratchpad memory. Additionally, labels required by the Completion Unit are initialized and directly passed to it. Threads do not pass packets to each other; each thread performs full ingress or egress processing only supported by the coprocessors mentioned above (run to completion).

Figure 2 shows the functional decomposition of processing a single packet [Ib02b]. Here, two DPPUs are depicted. Each DPPU consists of several rows and two columns. The columns represent the ingress and egress part of the processing. Each row illustrates a functional unit or a thread including time proceeding. Each CLP (core language processor) has two threads. CLP 0.1 and CLP 0.2 represent the threads of one CLP, and CLP 1.1 and CLP 1.2 of the other one. Between the two CLPs all coprocessors involved are listed. Each bar represents a period of usage; its pattern symbolizes the semantic functionality performed. The overall figure tries to qualitatively reproduce IP unicast fast-forwarding reference code. First, the TSE loads the initial packet data before the thread is invoked. During its execution, the thread employs several coprocessors: the CHK coprocessor verifies the packet header, one TSE performs longest-prefix match (LPM) on the upper half of the IP source address and the other computes a hash key. The thread concurrently updates header information such as Time-To-Live field and analyzes frame control information to direct its control flow. Possibly also a second LPM is launched, so that the result leaf of the first LPM is locked and unlocked with a semaphore. At the end of ingress processing, the packet is fetched and enqueued. Egress packet processing is also initialized and launches a TSE to perform a full match (FM) for an ARP lookup. Statistic counters are updated and finally the packet is enqueued to be put back onto the network.

The PowerNP utilizes data partitioning on packet level, a packet being the smallest PDU processed. Packet processing takes advantage of hardware assists supporting functional partitioning on the application level. A thread becomes a manager employing all these functionalities. Ingress and egress processing form a two-stage pipeline. Each pipeline stage is assigned to a thread. This architecture allows clusters of NPs interconnected by a switch fabric so that ingress and egress ports can reside on different nodes.

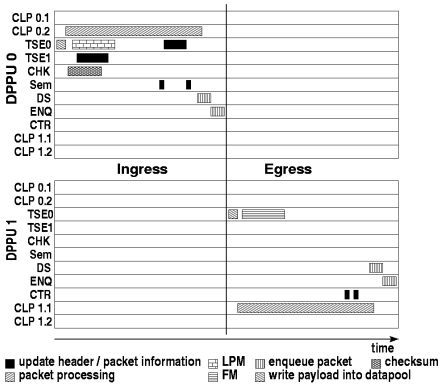

Figure 2: IP unicast packet flow of IBM PowerNP NP4GS3.

3.2 Intel IXP1200

The architecture of the IXP network processor [Ha99, In01c] includes six specialized packet engines, so-called microengines, for the data plane (Figure 1). A microengine consists of a processor core with hardware support for multithreading with up to 4 threads and reduced instruction set that provides only the functions needed for packet processing. Special functionalities built in hardware exist in the FBI (F-Bus Interface) Unit. It includes one coprocessor for generating adaptive polynomial hash keys and the IX Bus Interface for interconnection with the line interfaces. A scratchpad memory in the FBI Unit is shared on-chip, and two external memory segments, i.e. one SRAM and one SDRAM, can be accessed simultaneously by all six microengines, even for read and write operations. Furthermore, a StrongARM processor core exists to process control point tasks.

To demonstrate the packet flow for an IP unicast, Figure 3 shows the components involved and their functional activities to process a single packet. Here, we look at the Gigabit Ethernet Example Design [In01b] (reference code L3fwd2f) we used in our experimental setup. The IXP processes a packet in 64-byte segments (Mpackets). How an Mpacket is processed depends on its position in the Ethernet frame. The first Mpacket of the frame contains a SOP (Start of Packet) indication and the last Mpacket of the frame contains an EOP (End of Packet) indication. Any Mpacket in the middle of the frame contains neither a SOP nor an EOP indication. This indication is relayed from the GE MAC device, through the IX Bus Unit, to the receiving thread running in the microengine. Note that frames with length of 64 bytes or less are not segmented, and their Mpackets contain both SOP and EOP indications.

In Figure 3 we present the packet flow assuming a packet segmented in four Mpackets is received on port 8 (first GE port) and has to be forwarded to the outbound port 9 (second GE port). Each GE port is associated with two microengines for receive processing and one microengine for transmit processing. Therefore, a GE port has eight receive threads allocated to it, and each thread processes one Mpacket at a time. The receive thread processing the SOP Mpacket initiates processing of a new Ethernet frame buffered in the RFIFO of the IX Bus Unit. First, it allocates buffer in the datapool (SDRAM) and moves

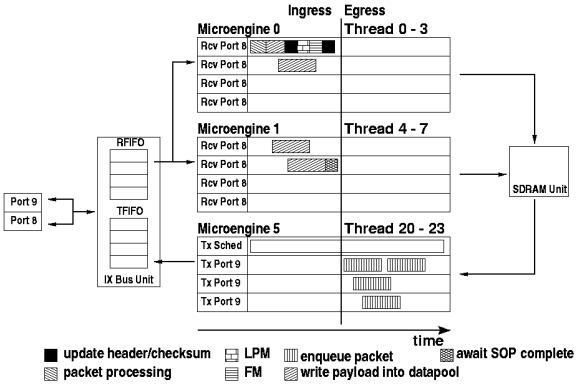

Figure 3: IP unicast packet flow of Intel IXP1200.

the payload of the Mpacket into this buffer. Then, it successively performs IP header updating (Time-To-Live field, checksum), a longest-prefix match (LPM), and a forwarding table lookup (FM) for retrieving the output port and the MAC address for the next hop. Finally, the new destination MAC address for the next hop is updated and the modified header is moved into the allocated buffer. A receive thread processing non-SOP Mpackets performs only the transport of the payload into the buffer in SDRAM. The receive thread processing the EOP Mpacket is also responsible for enqueuing the Ethernet frame onto the transmit queue. The thread that processes the SOP packet performs more computation and memory references than a thread that processes non-SOP Mpackets. Because of this the enqueueing thread has to wait for an indication that all header processing has been completed. Egress processing is established by one transmit microengine and fulfills packet data transport to a dedicated outbound port only. Here, one thread performs transmit scheduling by polling a scratchpad counter of pending packets in the transmit queue. This thread assigns Mpackets of a pending packet in the corresponding transmit queue to the remaining threads. These transmit threads move the Mpackets from the data pool in SDRAM into the TFIFO of the IX Bus Unit. The transmit thread moving the EOP Mpacket frees the allocated buffer in the datapool.
For this application the IXP employs functional pipelining only by decomposing ingress and egress processing into two stages performed by different microengines. To enhance throughput for GE forwarding, packet processing is performed by multiple entities of threads with identical instruction code. The processed PDU is an Mpacket of the packet. Thus, data partitioning is used as well. Note that the IXP network processor performs all functionalities by software. There is no hardware assist involved, such as a coprocessor.

3.3 Qualitative Comparison

Assigning any pure programming model to one of these network processors is rather difficult. Examining how the two architectures decompose data, their levels differ. Whereas the PowerNP's smallest PDU is a full packet, IXP decomposes all packets into Mpackets. Both NPs apply the same instruction code to all PDUs. All PDUs are processed

concurrently utilizing multiple threads on multiple processors. Pipelining leads one to expect a functional decomposition for which, see Figures 2 and 3, a two-stage pipelining is performed on packet level. Stages are ingress and egress. A further functional decomposition on the hardware level is only provided by the PowerNP. Employing special-purpose hardware such as coprocessors adds an additional dimension of decomposing tasks. Here, the instruction code performed utilizes asynchronous hardware functionalities and, thus, enlarges the level of concurrency.
Note that the NP platforms examined target different objectives. The PowerNP establishes an entire plug-and-play device which brings full router functionalities. In contrast, the IXP evaluation board only represents an experimental platform. The attached reference code for the microengines is not intended as a complete implementation of an IP router. It aims to demonstrate how to implement such a networking application on the IXP. Furthermore, the hardware designs of both network processors fulfill requirements of different levels in networking capacity. The PowerNP targets OC-48, whereas the IXP applies to OC-12 at maximum.

4 Application-Performance Measurements

4.1 Experimental Setup

To study and compare the performance of the two network processors, the PowerNP and the IXP are embedded into an existing cluster environment that is reduced here to two PCs, a Pentium III 866 MHz, 256 MB RAM, 32-bit-66-MHz-PCI bus, running Linux kernel version 2.4.10, and providing FastEthernet (FE) and GigabitEthernet (GE) MACs. We used GE based on optical fibers which is exclusively available for experimental purposes. In contrast to, for example, the Tolly Group test [To02] , here, latency and throughput are benchmarked on the application level and compared with a standard commercial switch that is frequently used in clusters. The benchmark application employed here is PerfDemo, a software for performance tests [Pe99] that quantifies application's throughput and latency. It is a parallel message-passing application using the communication library HPCC (High Performance Cluster Communication) [Gr98] and presumes a reliable interconnection between its processes. Underlying networking layers have to fulfill its qualifications. For each packet size, i.e. application's payload not including any protocol header, 100 bidirectional messages are sent to measure latency by computing half of the round-trip time. Throughput is determined by 1000 one-way messages. The applied protocol stack is itemized in descending order: HPCC, PVM (Parallel Virtual Machine) [Ge94], TCP (Transmission Control Protocol), IP (Internet Protocol) and GigabitEthernet via optical fiber/copper wire. Note that in general the number and size of messages on wire, i.e. on OSI level 1, differ from the application's values, i.e., OSI level 7, because of the protocol stack employed. The amount of additional traffic is rather low, only about one percent of packets sent through a link belongs to the daemons of HPCC and PVM.
To explore the application load behavior and performance of IXP and PowerNP different node connections are established. Measurements performed and presented in the following compare different connections: both nodes are connected (1) via PowerNP and media converters that are necessary to link optic and copper interfaces , (2) via IXP, (3) via a standard commercial GE-switch, (4) via a direct link, and (5) via media converters to measure their

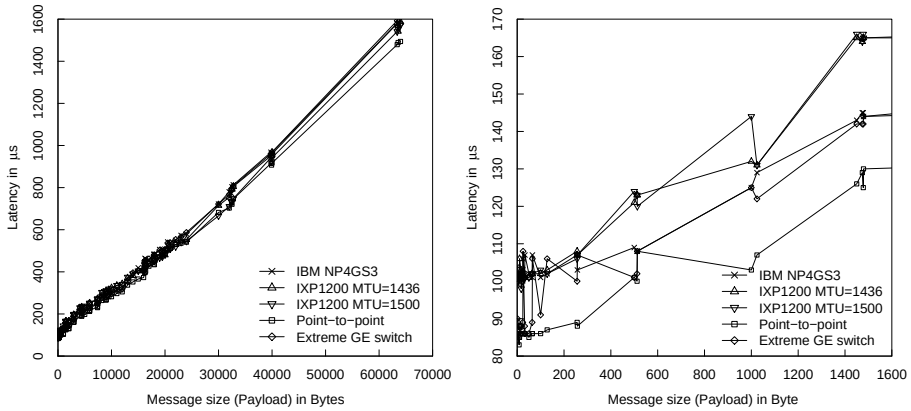

Figure 4: Measured latency, full (left) and MTU payload range (right).

impact on the application's latency and throughput. As we found no visible difference to the results of Direct Link measurements, further examinations will be omitted.
Therefore, the results presented here state any performance issues only regarding the experimental setup specified. The results were obtained for an application payload range of 1 to 64,000 bytes. MTU size is limited to 1436 bytes because the applied reference board of the PowerNP is restricted to this bound.

4.2 Experimental Results

To determine application-level throughput and latency, PerfDemo ran on all experimental setups. Here, we considered only measurements obtained by PerfDemo using buffered communication methods. The latency results measured on all kinds of connections are presented in Figure 4 showing the full payload range, and zoomed in on the payload range regarding the MTU size. We can see that the latencies in all kinds of connections differ only insignificantly. The measured latency ranges from 85 μs for small payload size to 1600 μs for full-sized application payload, i.e. 64,000 bytes. As expected, the direct link or point-to-point connection provides the lowest latency. Focusing on the results of payload sizes fitting into a single Ethernet frame, the GE switch and PowerNP perform best, consuming about 15 μs only for their additional hop. With the IXP latency is increased to about 35 μs for both MTU size configurations. The nonlinear trend of all curves is due to the buffer sizes implemented in the hosts, such as the hardware buffer in the GE NICs, and also the buffer used by the protocol stack. Packet aggregation on the TCP level also affects the system's overall latency.
Figure 5 shows the application-level throughput results. Clearly, 50 Mbytes/s is the best throughput of our node's hardware configuration performed with the direct-link connection. We can see that the measured level of throughput for all connections is in the range of 90% of this best-case performance. It is shown that the IXP reaches 48 Mbytes/s for a 1500-byte MTU size and 46 Mbytes/s for a 1436-byte MTU size. The GE switch and the

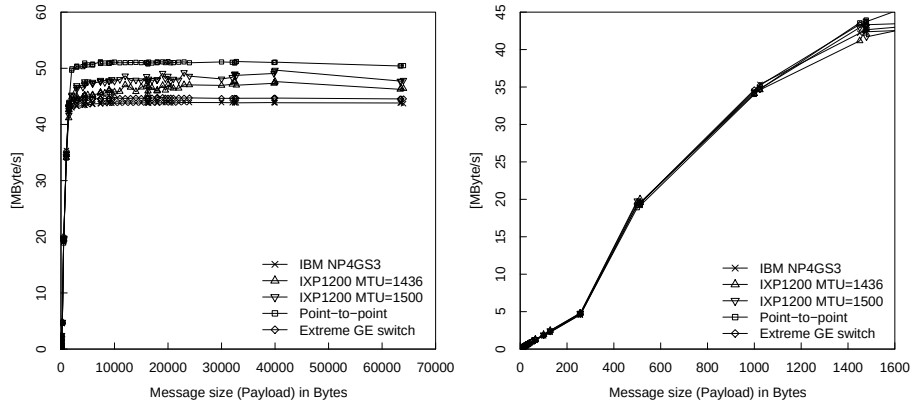

Figure 5: Measured throughput, full (left) and MTU payload range (right).

PowerNP have the same performance level at about 43-44 Mbytes/s[2]. This demonstrates the PowerNP can compete with the commercial GE switch. In respect to the throughput gap to the IXP, it has to be noted that the PowerNP provides a more comprehensive set of networking services. While the GE switch performs Layer-2 switching only, the PowerNP also establishes a Layer-3 router, i.e. more functionalities at the same performance cost level.
In our experimental setup, the IXP applies reference code that is limited to simple Layer-3 forwarding. Specific tasks of the networking domain such as ARP requests or ICMP messages are not supported, in contrast to the PowerNP. The evaluation board used presumes extensive software on the Control Point processor to which these tasks can be redirected and would be processed outside of the microengines. Owing to this offloading of functionalities, a simple comparison of the peek performance is not adequate. The observation that the IXP reaches a higher level of throughput than the PowerNP does could not yet be fully clarified and requires further investigations. It has to be examined in detail whether this results from the more complex packet buffering in the PowerNP or from unknown constraints different from that. Furthermore, the figures demonstrate the impact of the MTU size on throughput and latency. The IXP connection using the default MTU size shows a performance gain in comparison to the IXP connection using a MTU size of 1436 bytes. Note that this MTU-size limit is caused by the PowerNP's application reference board. It is expected that the PowerNP would benefit as well as the IXP of a larger MTU size.

5 Conclusion

In this paper we presented a comparative study of different network-processor architectures and their programming models, in particular the PowerNP and the IXP. These programming models, run to completion and pipelining, have been examined and their the-

[2]Note that the best-case throughput of our experimental setup is below the processing capabilities of the PowerNP, which has been proven to deliver OC-48c wire-speed performance by the Tolly Group test [To02].

oretical basics have been summarized. By a qualitative analysis of the processing of a single packet in respect to IPv4 forwarding, it is shown that the programming models implemented and architectures contain aspects of both programming models presented. By integrating both platforms into an existing GE cluster, the performance of PC-cluster communication is determined on the application level. Both platforms running available IP forwarding software can compete with a commercial, ASIC-based GE switch in terms of latency and throughput.

References

[Al03] J. Allen, B. Bass, C. Basso, R. Boivie, J. Calvignac, G. Davis, L. Frelechoux, M. Heddes, A. Herkersdorf, A. Kind, J. Logan, M. Peyravian, M. Rinaldi, R. Sabhikhi, M. Siegel and M. Waldvogel. PowerNP Network Processor Hardware Software and Applications, IBM Journal of Research and Development, Volume 47, Number 2/3, pp. 177-194, March/May 2003

[Ge94] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam. PVM: Parallel Virtual Machine – A Users' Guide and Tutorial for Networked Parallel Computing. The MIT Press, 1994

[Gr98] C. Grewe, W. Obelöer, S. Petri, and M. Boosten. Network Interface Software Development for Medical Applications. ESPRIT Project 20693 ARCHES Deliverable 3.4.1, Institute of Computer Engineering, University of Lübeck, Germany, December, 1998

[Ha99] T.R. Halfhill. Intel Network Processor Target Routers – IXP1200 Integrates Seven Cores for Multithreaded Packet Routing. Microprocessor Report Vol. 13, No. 12, September 13, 1999

[Ib00] IBM Corporation. IBM PowerNP NP4GS3 Network Processor, Product Brief, September 2000

[Ib02a] IBM Corporation. IBM PowerNP^{TM}NP4GS3 Network Processor, Datasheet, February 2002

[Ib02b] IBM Corporation. IBM PowerNP NP4GS3 – Advanced Software Offering Forwarding Picocode Design Reference, September 2002

[In01a] Intel Corporation. Intel IXP1200 Network Processor Family, Product Brief, Part Number 279040-001, 2001

[In01b] Intel Corporation. IXP1200 Network Processor Gigabit Ethernet Example Design, Application Note, Part Number 278407-001, August 2001

[In01c] Intel Corporation. IXP1200 Network Processor Family Hardware Reference Manual, Part Number 278303-008, August 2001

[Kr01] K. Krewell. Rainier Leads PowerNP Family – IBM's Chip Handles OC48 Today With a Clear Channel to OC192, Microprocessor Report Vol. 15, No. 1, pp. 10–12, January, 2001

[Pe99] S. Petri, G. Lustig, C. Grewe, R. Hagenau, W. Obelöer, M Boosten. Performance Comparison of Different High-Speed Networks with a Uniform Efficient Programming Interface. Scalable Coherent Interface - Proc. SCI Europe, 83-90, SINTEF Electronics and Cybernetics, Oslo 1999

[To02] The Tolly Group. IBM Corporation PowerNP NP4GS3 Network Processor IPv4 Forwarding Performance Evaluation, Test Summary, February 2002, No. 202111