Software processing performance in network processors

I. PAPAEFSTATHIOU[1], G. KORNAROS[2], N. ZERVOS[2]

1 Foundation of Research & Technology Hellas (FORTH),
Institute of Computer Science (ICS),
Vassilika Vouton, GR71110, Iraklio, Crete, GREECE
ygp@ics.forth.gr

2 Ellemedia Technologies
223, Siggrou Av,
GR17121, Athens, Greece
kornaros@ellemedia.com

Abstract

To meet the demand for higher performance, flexibility, and economy in today's state-of-the-art networks, an alternative to the ASICs that traditionally were used to implement packet-processing functions in hardware, called network processors (NPs), has emerged. In this paper, we briefly outline the architecture of such an innovative network processor aiming at the acceleration of protocol processing in high-speed network interfaces, and we use this architecture as a case study for our measurements. We focus on the performance of the general purpose processors used for executing high level protocol processing, since this part proves to be the bottleneck of the design. The performance is analyzed by executing a set of widely used, real applications and by applying network traffic according to certain stochastic criteria. The performance of the RISC used is compared with that of other well-known CPU architectures so as to verify that our results are applicable to the general network processors era. As our results demonstrate, the bottleneck of the majority of the network processors is the general-purpose processing units used, since today's network protocols need a great amount of high-level processing. On the other hand the specific purpose processors or co-processors, optimized for certain part of the network packet processing, involved in such systems, can provide the power needed, even at today's ultra high network speeds.

I. Introduction

The rapid growth in the number of network nodes, along with the ever-increasing users' demands for networking services, has imposed the development and deployment of high-capacity telecommunication systems. Such systems involve modules of high throughput, which often have their time critical functions implemented in application specific modules. The power required for the processing of protocol functions at wire speed is usually obtained either by generic microprocessors that are designed with the flexibility to perform a variety of functions, (at the cost of lower speed), or by Application Specific Integrated Circuits (ASICs) that are designed to meet specific functional requirements with high efficiency but with limited programmability. But the option that seems to be the more promising one today is a hybrid approach. This combines both chip technologies; a RISC processor as the central core, and ASICs to perform specific time-critical tasks. These components, called Network Processors (NPs), have been widely considered as the next generation standard platform for network processing.

While most of the NPs nowadays support the execution of higher layer protocol functions in software (such as routing, statistical compiling and reporting, error processing, connection admission control, network and transport layers protocol processing (e.g. ATM/AAL, TCP/IP, SSCOP) and traffic/resource management) the Programmable Protocol Processor or PRO3 outlined in section II, follows a different approach. The main target of PRO3 is to accelerate the execution of telecom protocols by extending a high-performance RISC core with programmable, pipelined hardware. CPU demanding and (hard) real-time protocol functions will be handled by the programmable hardware, while the remaining functions as well as higher layer protocols will be handled by the on-chip RISC, in an integrated way. The basic concept behind the PRO3 is to provide the required processing power through a novel architecture incorporating parallelism and pipelining, wherever possible, by integrating both fixed hardware and generic micro-programmed engines with general-purpose processors. However even by using all those dedicated, high-performance hardware modules, the bottleneck is the, very frequently underestimated, general-purpose processing, needed for the high level protocol handling. In particular, in the PRO3 there are dedicated units that perform the network-specific processing (like network header processing, packet classification, packet segmentation and reassembly, packet buffering etc), and leave to the general purpose processor, only the high level tasks (like simple mathematical calculations, and assignments of certain network data items to constant values based on another value carried in the packet). However, as this paper demonstrates, even those simple high-level tasks, are complicated enough so as to limit the performance of the whole NP. In other words, the bottleneck of the PRO3 architecture is the general purpose processing. We believe, that in the other existing NP architectures (like Intel's IXP[11] and Cisco's Toaster[12]), that have no dedicated units for network specific tasks, and all the processing is done in general purpose CPUs, this arguments will be strengthened.

The rest of the paper is organized as follows: Section II outlines the reference architecture, while Section III presents a performance analysis of the high level protocol

1530-1591/04 $20.00 (c) 2004 IEEE

processing on the PRO3 and this is compared with that of other architectures. Finally, section V concludes our analysis.

II. The PRO3 architecture

A trend in high-speed networking system design until recently has been to offload higher layer protocol functions, that are not performed at wire speed to external system control processors in a centralized manner. High level protocol processing such as: routing, statistical compiling and reporting, error processing, connection admission control, network and transport layers protocol processing (e.g. ATM/AAL, TCP/IP, SSCOP) and traffic and/or resource management, often today, are treated as a system's slow path. In order to accomplish the considerable processing power and memory throughput, required to execute protocol stacks for large numbers of connections, often more than one high performance processing units are employed. In such systems, the processing units are inadequate in supporting the protocol processing requirements for the entire set of active sessions. This constitutes a major system resource bottleneck, because the complexity of the protocol algorithms requires higher computational power than that offered by today's processor technology. Moreover, new services, applications and provider's requirements raise the significance of performing complex protocol processing tasks at ever increasing rates, since control and management plane functionality, as well as deep packet inspection and data transformation, are an integral part of modern telecommunication applications.

As mentioned above, the PRO3 system architecture, presented in this section, follows a different approach in the area of high speed protocol processing. The protocol processor comprises mainly of two distinct parts: a) The programmable, pipelined hardware modules that execute the CPU demanding and (hard) real-time protocol functions, and b) the high-performance RISC which handles the remaining functions, as well as the higher layer protocols. The concept of the PRO3 architecture is to provide the required processing power through a novel architecture incorporating parallelism and pipelining. Of key importance in this architecture is the integration of the processing elements of the system (mainly generic micro-programmed engines and RISC cores) with scheduler components, in order to facilitate data processing in a fair, balanced manner and to control data streams generated by the chip. When programmability is set as a major requirement, RISC based micro-engines are the best candidates for the implementation of functional units for many of the protocol processing functions. Following this approach we developed both fixed hardware units, as well as optimized micro-engines integrated with a commercial RISC processor in a layered architecture optimized for efficient protocol processing at link rates up to OC-48 (2,5 Gbit/sec). The actual block level PRO3 architecture is depicted in Figure 1. The PRO3 system is a distributed architecture incorporating dedicated hardwire modules for pre-processing and post-processing of low level protocols and two RISC-based Pipeline Modules (RPM), operating in parallel, to facilitate load balancing, as well as the execution of protocols with different incoming and outgoing data flows.

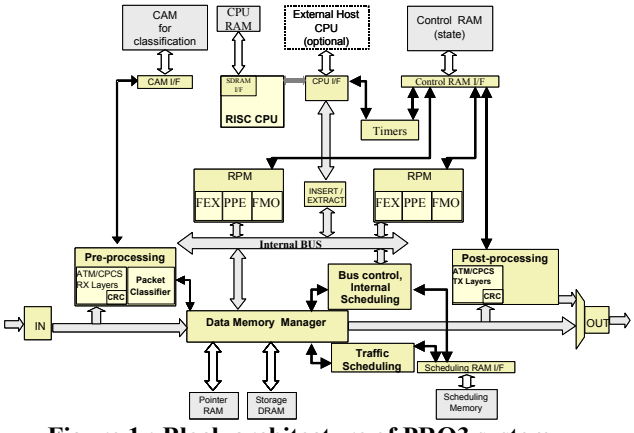

Figure 1 : Block architecture of PRO3 system.

In particular, the general purpose processing heart of the PRO3 system is the RPM [5], which consists of a modified RISC core (PPE) surrounded by a *Field Extraction* (FEX) programmable micro-engine, which directly loads the required protocol data to the RISC for processing, and a *Field Modification* programmable engine (FMO) for flexible PDU construction and header modification. All together form a powerful 3-stage pipeline module, which forms the mixed hardware and software processing heart of the system and it performs the main processing of each protocol.

The Internal Scheduling Unit [6], which is also a composite module, maintains a number of priority queues in order to schedule the forwarding of packets for processing according to the priority of each flow. It is also used to multiplex the execution of data transactions to the different internal destinations and/or allow for interleaved transactions over the Internal Bus. A dual scheduler module, configurable to operate either on fixed size cells or variable length packets, supports aggregate per group peak rate shaping for IP flows and guaranteed peak rate shaping per ATM flow.

Other main blocks perform data/queue management and higher layer protocol processing. The common high speed path (up to the transport layer) is performed in the PRO3 hardware pipeline, and higher layer applications on the internal Hyperstone RISC CPU. Packets are stored per-flow in the external DRAM in queues implemented as linked list data structures [2] and can be retrieved by the *Data Memory Manager* module (DMM)0 in response to specific commands. The packets are then delivered over the internal bus either to a) the RPM modules or b) the control RISC CPU or c) a host CPU (via the insert/extract interface) or d) the output interface.

In general, the following sequence of operations is applied to each incoming packet: reception, classification, state processing, and transmission. Each of these generic functions consists of a set of lower level functions and can be implemented in a different pipeline stage. In case of

exception the packet is redirected to the internal or the external CPU.

III. Performance Evaluation

In this section we will present the performance the PRO3 protocol processor can achieve when it executes real-world applications. It is worth noting that network processors constitute a new paradigm in network oriented computing architectures and as such no accepted benchmarking procedures exist [4], merely due to the polymorphism of architectures, as well as to the dissimilarity of applications. Our approach is based on running indicative computational intensive applications, which process network packets that belong to different network protocols.

The main processing power of the PRO3 comes from the two RPM units that operate in parallel, and the central RISC unit. RPM throughput is determined by the worst case performance of each of its pipeline stages and results are discussed in detail in this section. Of course, when wire-speed operation is achieved the performance is limited by the nominal speed of the interfaces of the chip. We have evaluated several applications and developed the PRO3 specifications to meet the performance targets that are included in 0. In the following analysis we show how these performance targets can be met. In particular, we concentrate on the IP applications.

The performance evaluation has been based on the following facts: the PRO3 chip is implemented using UMC 0.18 CMOS technology at a clock frequency of 200MHz and with a 64-bit-wide internal bus. As it has been briefly described above, the memory bottleneck has been coped with by paralelizing memory accesses per functional unit so as each one of them can get the required bandwidth even under worst case conditions.

For the IP applications the worst case conditions arise when there is a continuous stream of minimum size 40-Byte IP packets. In this case each RPM stage has about 256 ns for processing each packet. This number comes from the fact that the network interfaces operate at 2.5Gb/sec and the load is balanced between the two RPMs, which operate in parallel.

Table A PRO3 features

	Application	Sustained rate	Max flows
ATM applications			
1	ATM cell processing	2,5 Gbps	512K
2	AAL5 processing	2,5 Gbps	512K
IP applications			
3	Layer 2, 3, 4 classification	2,5 Gbps	512K
4	Layer 2, 3, 4 filtering	≤2,5 Gbps	512K
5	Layer 4 stateful inspection	≤2,5 Gbps	512K
6	NAT	≤2,5 Gbps	512K

The applications used throughout this section implements a *stateful inspection Firewall* with *Network Address Translation* (NAT) support [3], [5]. Samples of real TCP/IP traffic have been used as input in the H/W simulation and the processing time in the Modified Hyperstone RISC processor (MHY) was accurately measured. In order to evaluate the application performance, experiments with different packet lengths and packets of different protocols were carried out. The main parameters measured were the throughput these modules can achieve and the total number of the executed instructions under the different circumstances. Our analysis is focused on the RISC processor only since as it is presented in [8] the other RPM modules can always achieve the performance of Table A. In order to show that the problems associated with software processing is not due to the specific RISC core we use, we have also measured the performance, in the same context, of two of the most widely used general purpose CPU families: the SPARC and the Intel x86.

In general MHY's throughput depends heavily on the custom running application and it is estimated, that for complex applications, like TCP state updating and NAT, less than 170 instructions are needed (notice that the core is optimised for code density as it is described in [5]). But as it will be shown, even this small number of instructions can cause significant problems in the performance of the overall design.

As 0 demonstrates the number of instructions executed on the MHY depends on the protocol of the packet as well as on the value of the packet header and obviously the specific application running.

Table B : Number of Instructions for different protocols/applications

	NAT	No-NAT, packet passed	No-NAT, packet dropped
TCP	141 – 164	60 – 92	42 – 68
UDP	95 – 107	36	N/A
ICMP	N/A	36	N/A

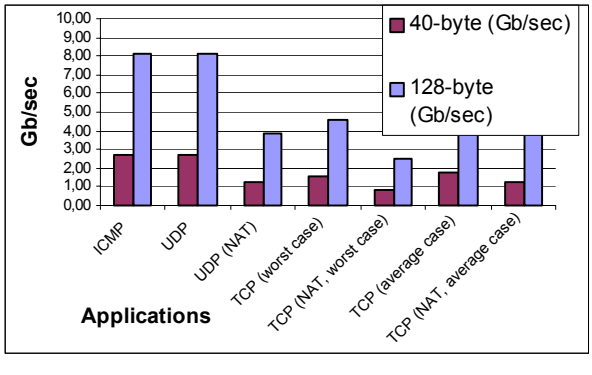

Figure 2 : Throughput achieved for different applications

One of the main characteristics of these applications (and the higher layer protocol applications in general) is that they process only the headers of the packets. As a result, their performance depends on the size of the packets (since if the packet is small they will have to process a lot of headers at a short time, whereas in case of large packets fewer headers should be processed). In Figure 2 the throughput achieved for different packet sizes is demonstrated. As it is clearly shown if the network streams comprises of packets with an average length of 128 bytes (which is less than the typical mean IP packet length) the two MHYs can process the packets at the requested rate (2.5 Gb/sec). It is worth noting that the packet classification, queuing and scheduling elements can support 2,5Gbps link rates even for worst case minimum packets. It should also be stressed that the average cases in TCP and TCP-NAT processing have been derived as follows:

We have first analysed millions of packets from the backbones of various network providers in the U.S.A. [9] so as to derive the percentages of packet belonging to each particular TCP state. Then we plugged these percentages in our experiments and we figured out what would have been the average number of instructions needed if these real network traces were sent over the PRO3.

In order to investigate whether the low bandwidth achieved in case of small packages, is due to the specific RISC-core used, the same application has been compiled into different CPU architectures. In particular, we have chosen two of the most representative architectures: the SPARC and the Intel x86 one. As it is known, the SPARC is RISC while the Intel x86 is CISC. The number of instructions needed in each CPU (in the worst case) is shown in Figure 3. The numbers where produced by applying the highest compiler optimisations for code density and handcraft the produced code so as to further reduce the number of instructions needed.

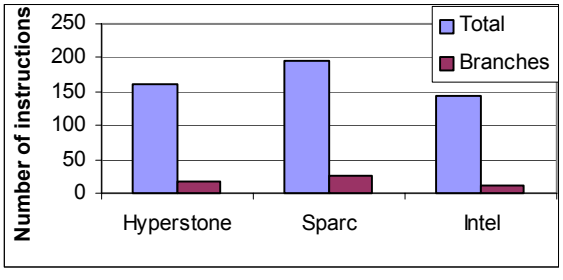

Figure 3 : Number of instructions in different architectures

This Figure demonstrates that the Hyperstone instructions needed are not significantly more than the x86 ones and less than those at the SPARC. Since x86 is a CISC architecture we would expect that it would have needed less and more complicated instructions. Due to the fact that in every architecture we need at least 145 instructions to implement TCP stateful inspection and NAT we support that the software processing time can cause significant problems in the majority of the Network Processors that are organized around a general CPU core, and not only on PRO3.

Since in most of the pipeline CPUs today the performance of a program degrades with the number of branches (due to pipeline stalls/flashes) we have also counted the number of branches for each architecture (as shown in Figure 3). It is clear that the percentage of branches does not vary significantly in the measured architectures, even though the actual number of branches varies (This is because in the Intel Architecture we needed far less instructions). In general, based on the results presented in Figure 3, we claim that even if we have changed the particular CPU we use with another general CPU (either CISC or RISC), we would have not done much better in terms of the software performance.

As it has been mentioned above the performance of the software depends on the size of the packets as well as the protocol the network traffic belongs to. Therefore, we have created a random mix of packet sizes and network protocols and we have measured the performance of the software when processing these traces. The results for the average cases, shown in Figure 4, demonstrate the bandwidth achieved when the percentage of the small packets in the mix changes. The small packages are those with length less than 100 bytes. The traffic generator produces both small and large packets based on a Gaussian distribution with a mean of 70 in the small packet's case[1] and 170 for the large packets.

As the figure clearly demonstrates even when the flow consists of 90% of small packets the software can process the data at the requested speed so as to achieve the 2.5Gbps bandwidth of the PRO3.

Unfortunately, this is not the case when all the packets belong to the worst-case scenario. Even though, this scenario is very unlikely to occur in the real world, for reasons of completeness, we also demonstrate the performance of the software when each of the packets (irrespective of their sizes) need 167 instructions so as to be processed.

As Figure 5 shows, in order for the software modules to have the time to process the packets at the peak rate of the PRO3, the mix should include at least 50% of packets with length greater than 100 bytes. Measurements from real networks show that in a typical real-world network at least 50% of the packets are indeed greater that 100 bytes [9]. In particular, as [9] clearly demonstrates, about 30% of the TCP traffic consists of 40 bytes packets whereas the rest of the packets are significantly larger. In the experiment results of 5, we have tried to create short packets as close to the statistics measured at real networks, as possible. As a result, someone can argue that even if all the packets trigger our worst case, the PRO3 can process them as long as they have lengths similar to those in a typical IP network.

Another interesting aspect of the software performance is how it varies with the network protocol the packets belong

[1] All the packets are longer than 40 bytes according to the TCP protocol's specifications[4].

to. Figure 6 demonstrates the bandwidth achieved when a network stream of TCP and UDP packets is processed.

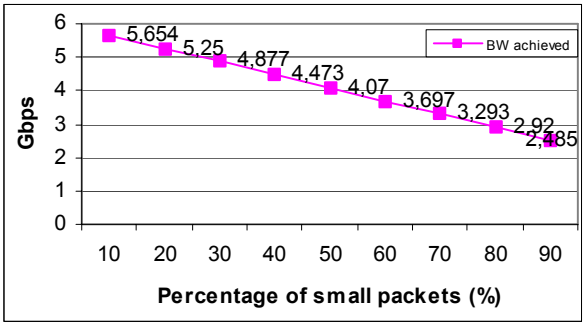

Figure 4 : Bandwidth achieved for a traffic mix in the average case

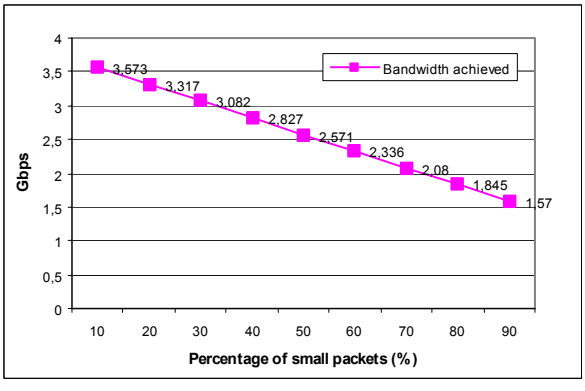

Figure 5 : Bandwidth achieved for a traffic mix in the worst case

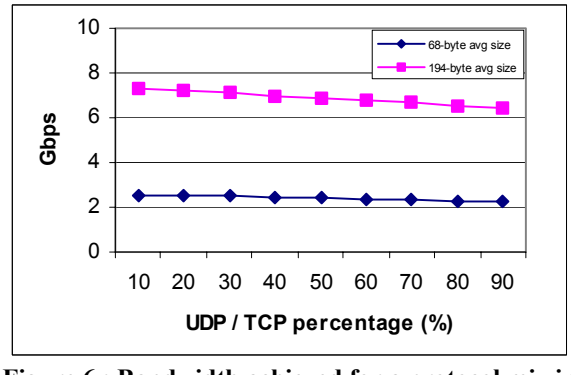

Figure 6 : Bandwidth achieved for a protocol mix in the average case

It is clear that the two MHY of the PRO3 can process the packets at peak rate when more than 20% of the packets are UDP packets. Since in a real network stream we have at least 20% of UDP traffic, we believe that even when the network packets are small (mean length of 69 bytes) the PRO3 can satisfy the bandwidth requirements of the interface circuits. Obviously this argument is further strengthen for the long packets case. In the experiments that produced this graph, the average TCP NAT case scenario was used. Obviously if all the TCP packets trigger the worst case processing, the number of packets the two MHY can process will be lower and so would be the bandwidth the whole system can provide.

Finally, it should be noted that we have chosen not to present, in this paper, the performance of the processing engines of other NPs, since those engines are very simple and according to our experiments [10], they need much more instructions than the presented processors, for executing the same network applications. Therefore, we claim that even if it were possible to use powerful microprocessors (like an x86, or a SPARC), within an NP (which is not feasible in today's technologies due to area and power limitations), the design's bottleneck would probably still be the general-purpose high-level processing.

IV. Conclusions

In this paper the Programmable Protocol Processor (PRO3) was outlined with emphasis in its performance when executes real-world network applications. The performance of the software processing of the system is analysed and ways of optimising it proposed. The performance is also compared to that of other software processing systems. The results show that in the current (and probably future) high-speed networks, the general-purpose software processing elements would be the bottleneck, and not the hardware modules that are optimized for packet processing. The PRO3 chip has been fabricated in UMC's 0.18 μm logic process occupying about 37mm^2 of area, and consisting of 856K gates and 1Mbit of on-chip SRAM.

V. References

[1] A. Nikologiannis, M. Katevenis, "Efficient Per-Flow Queueing in DRAM at OC-192 Line Rate using Out-of-Order Execution Techniques", *in proc. of ICC2001*, Helsinki, Finland, June 2001.

[2] N. Nikolaou, J. Sanchez, T. Orphanoudakis, D. Polatos N. Zervos, "Application Decomposition for High-Speed Network Processing Platforms", 2nd European Conference on Universal Multiservice Networks, ECUMN'2002 April 2000, Colmar France.

[3] EDN Embedded Microprocessor Benchmark Consortium (EEMBC) "Network Processing Platform Benchmarking Methodology Framework", *Draft 1.0 Request for comments*, July 2000.

[4] Guido van Rooij. "Real Stateful TCP Packet Filtering in IP Filter," White paper.

[5] G. Lykakis et al. "Efficient Field Processing Cores in an Innovative Protocol Processor System-on-Chip", Design, Automation and Test in Europe 2003 (DATE 2003), Munich, Germany 3-7 March, 2003.

[6] G. Kornaros, I. Papaefstathiou, A. Nikologiannis, "A Fully-Programmable Memory Management System Supporting Queue Handling at Multi Gigabit rates", In Proceedings of the 40th IEEE/ACM Design Automation Conference (DAC), Anaheim, California, U.S.A., June 2-6, 2003.

[7] I. Papaefstathiou, et. al. “An Innovative Scheduling Scheme For High Speed Network Processors”, 2003 IEEE International Symposium on Circuits and Systems (ISCAS’03), Bangkok, Thailand, May 25-28 2003.

[8] K.Vlachos et.al. ““Processing and scheduling components in an innovative network processor architecture”, to appear, in proc. VLSI Conference 2003, New Delhi, India, January 2003.

[9] A.J. McGregor, H-W.Braun and J.A. Brown, “The NLANR Network Analysis Infrastructure,” IEEE Communications Magazine, Vol. 38 (5): pp. 122-128, May 2000.

[10] I. Papaefstathiou et.al. “A Network SoC for Embedded Protocol Processing in Multi-Gigabit Networks”, submitted to IEEE Micro.

[11] “The Next Generation of Intel IXP Network Processors”, Intel Technology Journal, Volume 6 Issue 3, August 2002

[12] “Cisco Inc. Cisco's Toaster 2 Chip Receives the Microprocessor Report Analyst's' Choice 2001”, Award for Best Network Processor. http://www.cisco.com