Packet Processing Acceleration With a 3-Stage Programmable Pipeline Engine

I. Papaefstathiou, K. Vlachos, N. Nikolaou, N. Zervos, and V. B. Lawrence, *Member, IEEE*

***Abstract*—In this letter, we present the architecture and implementation of a novel, 3-stage processing engine, suitable for deep packet processing in high-speed networks. The engine, which has been fabricated as part of a network processor, comprises of a typical RISC core and programmable hardware. To assess the performance of the engine, experiments with packets of various lengths have been performed and compared against the IXP1200 network processor. The comparison has revealed that for the case study shown in this letter, the proposed packet-processing engine is up to three times faster. Moreover, the engine is simple to be fabricated, less expensive than the corresponding hardware cores of IXP1200 and can be easily programmed for different networking applications.**

***Index Terms*—ASIC, Network Processor, Special Purpose Processor.**

I. Introduction

With the rapid growth of Internet traffic and the increasing line rates, the execution of the various networking tasks is increasingly considered to be the main bottleneck for communications. To meet the stringent processing demands, designers are faced with two alternatives: either create a custom hardware solution (ASIC) or use a special purpose processor, called network processor (NP). The ASIC approach can achieve the desired speeds, but it is inflexible, since changes in the functionality are very limited or not permitted at all. However, since protocols continue to evolve, accommodating new features that comply with the latest standards is of significant importance. To this respect, NPs can provide the required flexibility and programmability.

In this letter, we present a flexible and programmable engine that can sustain wire speed protocol processing, even for complex and high demanding networking tasks. The design can be easily embedded in any networking environment (i.e., both ASICs and NPs). It combines a typical RISC core [1] with custom-made, fully programmable hardware in a 3-stage pipeline module. In this way, the efficiency of a typical CPU is enhanced by providing the means to tailor its circuits for special tasks and, reversely, the application diversity of highly optimized hardware is significantly broadened. Using this engine, that incorporates a low cost and simple general purpose RISC, at 200 MHz, we were able to sustain stateful inspection firewall processing and Network Address Translation (NAT) for 2.5 Gb/s TCP/IP traffic.

Manuscript received June 11, 2003. The associate editor coordinating the review of this letter and approving it for publication was Prof. K. Park.

I. Papaefstathiou, N. Nikolaou, and N. Zervos are with Ellemedia Technologies, Athens GR17121, Greece (e-mail: yanni@ellemedia.com; nikolaou@ellemedia.com; nzervos@ellemedia.com).

K. Vlachos was with Bell Laboratories Advance Technology EMEA, Lucent Technologies, 1200BD Hilversum, The Netherlands.

V. B. Lawrence is with Bell Labs, Lucent Technologies, Holmdel, NJ 07733 USA (e-mail: vbl@lucent.com).

Digital Object Identifier 10.1109/LCOMM.2004.823427

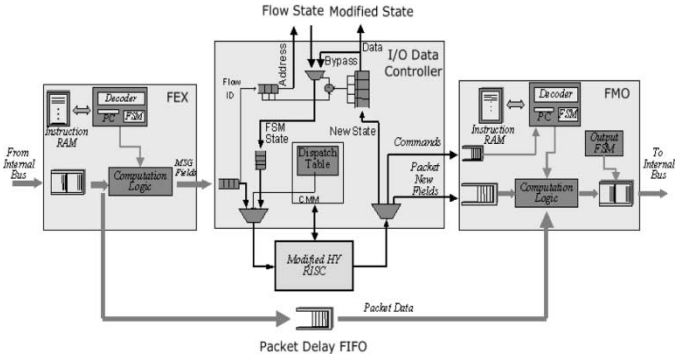

Fig. 1. Programmable processing functional model and block diagram.

II. The Programmable Processing Engine—PPE

The Programmable Processing Engine (PPE) (see Fig. 1) is a 3-stage pipeline module, consisting of three logical sub-units: a Field Extractor (FEX) unit, a typical RISC core and a Field Modification unit (FMO). More particularly, programmable hardware is commissioned to extract fields from incoming packets and feed them to the processing core. After the fields' processing in the RISC core, FMO updates, in a programmable manner, specific fields of the packet. Additionally, an I/O data controller is used to relieve the processing core from *I/O* duties and free available resources for real processing.

The Field Extraction operation is controlled by microcode, stored in an internal SRAM. The instruction set comprises of simple and generic instructions that operate over data stored in a FIFO of 32-bit words. The FEX instruction set supports the following operations: 1) variable length (1 to 32 bits) field extraction; 2) backward/forward movement in the data FIFO; 3) conditional jumps; and 4) addition. FEX instructions are flexible enough to allow conditional branches based on the content of extracted filed (e.g., protocol field of the IP header), as well as parsing of protocol headers based on header and packet length information (e.g., FEX can be easily programmed to recognize and extract or skip IP and TCP options). The execution time of the field extraction operation is constant and does not depend on the number of extracted bits (only on the number of the fields extracted).

Packet processing is initiated by a packet arrival at the FEX input interface. After the field extraction, the I/O Data Controller places the extracted fields directly to the register file of the RISC core. In this way the RISC performance is significantly enhanced, as I/O operations are performed in parallel with the

processing of the previous packet. When the RISC processing ends, the I/O controller hands over the processed fields from the core, as well as a set of commands to FMO. The operations of FMO, which is also controlled by microcode, include field extraction and field modification. The FMO's instruction set is very similar to the FEX's one.

In general, the FMO can be used to modify a packet or create a new packet (using a packet template that is hard-coded in the firmware, plus new fields received from the RISC core). Additionally, if the firmware is large enough it can generate a large number of new packets.

Another important component of the PPE design is the Packet Delay FIFO (see Fig. 1). The received data (packet or part of a packet) are temporarily stored in the Delay FIFO waiting for fields/packet processing results. When the FMO receives the results (new fields and commands) from the Data Controller, then depending on the processing result and the application, it may modify or delete the packet/part of packet in the FIFO, or construct a new one, under the control of the firmware.

Regarding the state-of-the-art in network processing units, the approach followed by the majority of the communication systems' manufacturers (e.g., in Agere's PayloadPlus [3], IBM's PowerNP [4], Intel's IXP 1200 [6], Motorola's C-5 [2]), is mainly a brute force, where a large number of general purpose RISC engines are plugged together. Those engines are connected via either point-to-point links (IXP, PayloadPlus) or shared high bandwidth common buses, or centralized network topologies (PowerNP, C-5). There are also simple, dedicated, hardware units for performing certain complex tasks in hardware. However, the processing units have no special hardware for optimizing the data transfers between them, or certain instructions that increase the speed of the network data processing (except of the PowerNP, which has a unit similar to our FEX but it does not have any units with the FMO functionality).

The main feature that differentiates our approach from that used in the abovementioned Network Processors and contributes to the higher cost-performance effectiveness of the PPE, is the highly sophisticated interconnection mechanism of the RISC with the FEX and the FMO. In particular, the register file of the RISC is divided into two parts, and while the core processes the data on one part, the Data Controller (specialized hardware) writes or reads new fields, coming from the FEX or going to the PPE, to and from the other part of the register file. In this way, the data processing can take place in parallel with the data movements. Therefore, the RISC is not stalled at any time, whereas the processing units in the existing NPs should perform the data movements themselves.

Similarly, the cache of the RISC has an external port, and therefore the state of the network flows is written to it without the need of any RISC command. The RISC processes the data directly from its cache and writes the updated state back to it. The architecture of both the FEX and the FMO allows optimal field processing and modification. Therefore, these engines need fewer commands than the general purpose RISC engines used in other NPs in order to perform the same tasks. IBM's power NP is the only one which seems to follow a similar approach by having special processing units for field extraction, but its performance as shown in [4] seems to be lower than that of the PPE (demonstrated in Section III).

Both FEX and FMO have an extremely low hardware complexity, since they have been designed for only field extraction/insertion and modification. The gate count of the two engines (about 50 K) is similar to an implementation of a very simple 16-bit microcontroller (such as in [7]), when at the same time, in the network applications, they are more effective than many 32-bits general purpose CPUs.

III. Performance Evaluation

The performance evaluation of the PPE has been based on the following facts: implementation using UMC 0.18 CMOS technology, 200-MHz clock speed, and 64-bit-wide input/output interface. Real TCP/IP traffic has been used as input and the number of executed instructions of each block was measured. However, overall performance depends heavily on the instructions executed on the RISC core, which in turns depends on the application. In our case study, a TCP stateful inspection with NAT was used [5]. Fig. 2(a) displays the fields that are extracted and are fetched for processing to the core, via the I/O data controller, while Fig. 2(b) and (c) displays the number of executed instructions for the FEX and FMO engines, respectively.

In the case of FEX, from Fig. 2(b), we can observe that for small IP packets (up to 64 bytes) the number of required instructions is proportional to the IP header, which depends on the IP options length. Additionally, for IP packets larger than 64 bytes, a fixed number of instructions are required for the cases where the IP header length is between 20 and 48 bytes, while for the rest of the IP header length cases, this number increases proportionally to the IP header length, reaching a maximum of 43 instructions in case of an IP-packet with a 60-byte IP header. On the contrary in the case of FMO, the number of required instructions grows linearly with the IP header length. This is because, in the case of header field modification, the header data are sequentially scanned in order to reinsert the updated fields in the correct slots. Fig. 2(c) displays the case of field modification for small IP packets (left column), as well as for large packets (right column).

Table I demonstrates the performance of the whole PPE engine when executes a TCP stateful inspection application with NAT support. The PPE performance is compared with that of the Intel IXP1200 [6]. In order to perform the comparison, we have ported our application to the IXP1200 and we have used three of its microengines; executing the FEX, the FMO and the RISC program, respectively. This organization proved to be the optimal one for the IXP due to the fact that the processing was almost perfectly balanced between the three microengines. This was possible because the FEX, FMO and PPE programs have all a similar complexity. Additionally, because all the data needed were in the IXP's local scratch memory and there were no long time periods that the microengines were stalled due to external accesses, the multithreading commands of the IXP could not increase the processing bandwidth. Moreover, plugging the same programs to more microengines proved to decrease the performance of the system since the data transfers needed between the microengines imposed a significant processing overhead. The instruction count numbers for the IXP1200 where produced by

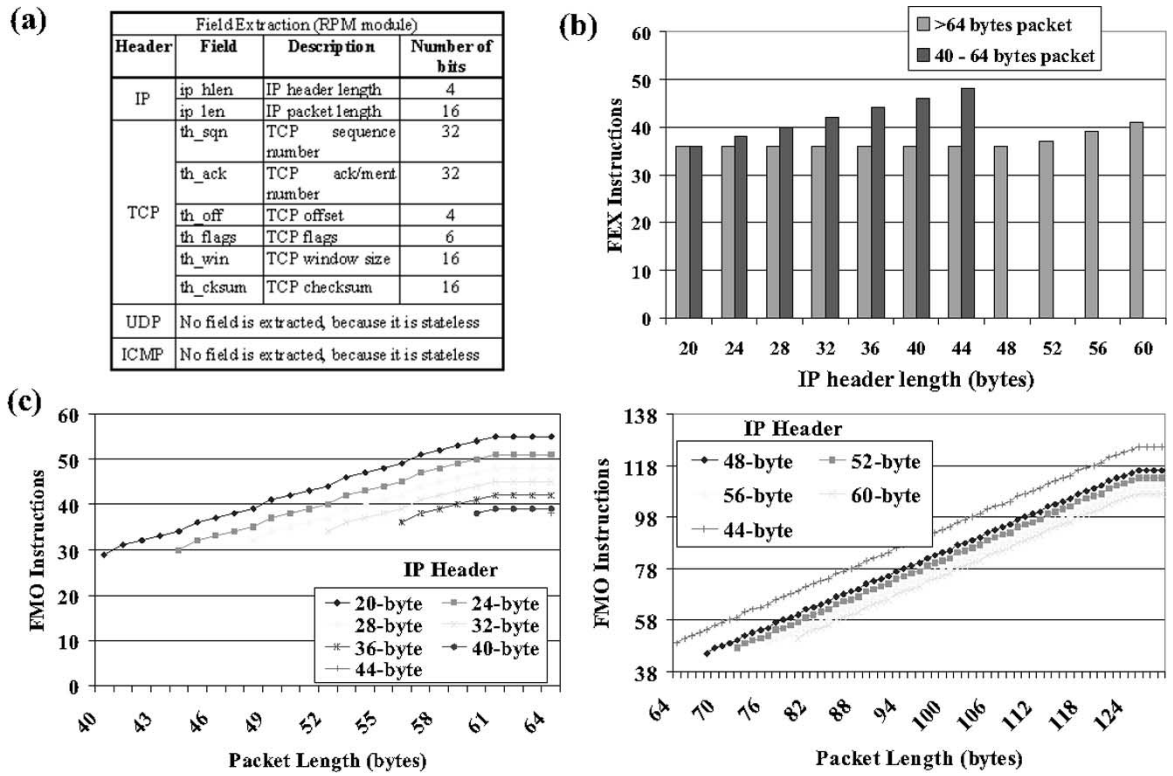

Fig. 2. Fields extracted by FEX and number of instructions and processing time for the FEX/FMO.

TABLE I
MEAN INSTRUCTION COUNT & LATENCY FOR PPE/IXP1200

	TCP NAT (delay in ns)	UDP no-NAT (delay in ns)	TCP NAT + UDP noNAT (delay in ns)
PPE	286 (1770)	193(1285)	472 (2980)
IXP1200	900 (3870), 3μengines	435 (1870), 2μengines	1071(4605), 5μengines

applying the highest compiler optimizations for code density and handcrafted the produced code, so as to further increase the performance. Those results of Table I clearly demonstrate that in such applications the PPE is up to 2.5 times faster (and at least 50% more efficient) than the IXP1200, even though the IXP1200 runs at 233 MHz. It must be noted that in the IXP1200 case we used up to five microengines (when processing both a UDP and a TCP packet at the same time) and even in this case the performance of the PPE is extremely higher. Additionally, the FEX and FMOs implementations are much smaller than the corresponding microengines.

In general, we believe that the PPE, due to its unique features, can do the protocol processing at a much higher rate than the corresponding cores found in the current NPs (including the IXP1200) since the latter have barely no special hardware for data transfers or special commands for data extraction/modification. Moreover, our PPE has a much lower hardware complexity, (for example, five times less hardware is needed for the PPE implementation than the Motorolla's corresponding 3-channel processors [2]), making it much more cost-effective for state-of-the-art NPs or networking ASICs.

IV. CONCLUSION

In this letter, the architecture of the PPE was presented and its performance has been assessed when executing a stateful inspection with NAT support. The PPE uses an innovative concept of a 3-stage pipelined module, integrating a RISC core with special purpose programmable hardware on the same processing platform. In this way, the advantages of a general purpose CPU are combined with those of special purpose (networking) processing cores, in the most efficient manner. The demonstrated results verify that the PPE is significantly efficient in such an application. The design is suitable for both cell and packet based network-processing applications and can be embedded in any processing environment. The PPE has been fabricated within a highly sophisticated network processor at UMC's 0.18 μm logic process, occupying about 9.2 mm^2 of area (of which about 6.5 mm^2 are covered by the RISC core and its associated memories) and working at 200 MHz.

REFERENCES

[1] Hyperstone Electronics E1-32X RISC/DSP [Online]. Available: http://www.hyperstone.com
[2] (2002, June) C-5 Data Sheet. Motorola. [Online] http://www.motorola.com
[3] (2003) Agere Systems, Advanced Payload Plus Data Sheet. [Online] http://www.agere.com
[4] R. Allen *et al.*, "IBM powerNP hardware software and applications," *IBM J. Res. Develop.*, vol. 47, no. 2/3, pp. 177–194, Mar./May 2003.
[5] G. Rooij, "Real stateful TCP packet filtering in IP filter," presented at the 2nd Int. SANE Conf., Maastricht, The Netherlands, May 22–25, 2000.
[6] M. Adiletta *et al.*, "The next generation of Intel IXP Network processors," *Intel Technol. J.*, vol. 6, no. 3, Aug. 2002.
[7] B. Finch and W. Miller, "A new reference design development environment for JPEG 2000 applications," presented at the *System-on-Chip and ASIC Design Conf.*, Jan. 2003.