A Quantitative Study on Layer-2 Packet Processing on a General Purpose Processor

Mostafa E. Salehi, Ramin Rafati, Farshad Baharvand and Sied Mehdi Fakhraie
School of Electrical and Computer Engineering,
University of Tehran, Tehran 14395-515, Iran
Email: {mersali, rrafati, baharvand, fakhraie}@ut.ac. ir

Abstract— In this paper, we present a quantitative study that investigates implementation of a layer-2 switching application on a general purpose processor (GPP). The objective is to better understand the main challenges and tradeoffs in using such processors for packet processing applications. The goal of this study is to identify the architectural guidelines for successful development of an application specific instruction processor (ASIP) for such applications.

To asses the performance of switching of packets with various lengths, a LEON2 RISC processor has been chosen as a GPP. The obtained results are compared together based on detailed instruction level profiling of the mentioned application.

***Index Terms*—Layer-2 switching, packet processing, network processors**

I. Introduction

THE rapid evolution of optical networking technology, and hence, the increased capacity of physical interconnection media leads network system users to consider the bandwidth as an always-available resource. This has shifted the network bottleneck to the execution of various networking tasks, therefore forcing the gradual replacement of processing devices with more powerful engines. As a result, there has been tremendous interest in speeding these processing nodes, making the equipment run faster by means of specialized chips to handle data trafficking.

To meet the processing demands of the network applications, NPs target the area between flexibility and high data rates extremes. NPs are the response to the need of cost effective, yet flexible system solutions for evolving applications that allow packet processing at high data rates. They can be reprogrammed while they are deployed in a network and also accommodate upgrades, protocol changes, or bug fixes. In the best case, this leads to a longer in-use duration for NP solutions over their ASIC counterparts.

Furthermore, many NPs incorporate hardware micro-engines and packet processors resembling behavior of an optimized instruction set RISC microprocessor. Packet processors are high-performance, programmable devices with special architectural features that are optimized for network packet processing. They are mostly embedded within network routers and switches and are designed to implement complex packet processing tasks at high line speeds. Thus, packet processors support the execution of network-specific functions at very high speeds, while retaining flexibility due to their programmability nature.

The flexibility and lower development time has raised considerable interests in network processors within the commercial communications sector. Today's commercial network processors are optimized for packet header processing by deploying custom packet processor engines. The Intel IXP1200 [1], IXP1250 [2] uses six micro-engines, IBM PowerNP [3] uses 8 processing units, and Motorola C-5 [4] uses 16 processing units on the same chip.

In this paper we are using a GPP to handle the entire tasks of a switching application. Using GPPs provides maximum flexibility but incurs major challenges in meeting required processing performance. The results presented in the following sections indicate what are the bottlenecks in the way of achieving processing requirements of the packet processing when GPPs are used as the sole processing engine. We are using these results to offer an optimized micro-engine to be embedded in the most networking tasks in our ongoing works.

Section II describes the layer-2 switching (L2S) flow and the tasks of typical switching software. In Section III we present the implementation of the L2S software on LEON2 processor. In Section IV the simulation results of L2S software on LEON2 processor are presented. Our quantitative results are based on an instruction level profiling. Section V presents the proposed architectural guidelines for a high performance packet processor, and Section VI concludes the paper.

II. Layer-2 Switching (L2S) Flow

The basic task of the L2S is to direct frames from source MAC to destination MAC(s). DMA engine of the MAC transfers frame from external port to the memory (receive) and from memory to the external port (transmit). Associated with each MAC there is a control field which is called Buffer Descriptor (BD). BD contains a pointer to the actual frame location in the memory along with its length and various information regarding type of frame such as broadcast, multicast, VLAN, priority, and etc. Software is responsible to initialize BDs and then process them during MAC operation.

The main flow starts by initializing software related variables, MAC registers, BDs, frame memory, forwarding table, and other components of the system. Forwarding table is implemented in a hash mechanism and contains the MAC address–port number pair. After the initialization, MAC will be enabled so that frames can be received from the Ethernet ports.

The entire tasks of typical switching software can be summarized as follow.

1. MAC and BD initializations
2. Setting proper configuration
3. Receiving frames
4. Extracting frames borders
5. Releasing memory of erroneous frames
6. Applying Ethernet specific rules (detecting broadcast frames, priority fields, …)
7. Applying specific filtering rules (MAC or port based filtering, VLAN segregation, …)
8. Applying switching specific rules (hash table lookup)
9. Preparing transmit BD
10. Enabling destination MAC to start transmission
11. Communicating with layer 3
12. Preparing BDs for new reception
13. Dealing with various error conditions (TX/RX buffer overflow, MAC errors, …)
14. Keeping statistical information from switching status

After the initialization task which is performed by *L2S_init(),* the *L2S_process()* waits for a frame to be received. Upon receiving a frame the *Frame_process()* is called and it has the responsibility to extract frame border. It is possible for a frame to be received incompletely because of the CRC error, buffer overrun, and etc. In these cases MAC does not recover the occupied memory of the system and software should release the memory for future receive operations.

Upon receiving a complete frame, *eth_process()* is invoked. Initially, it stores source address in the hash table (*l2ht_learn()*) and if frame is not a broadcast one, it searches for destination address within hash table (*l2ht_search()*). If the address is found, frame will be transferred directly to the designated MAC (*frame_transmit()*). Otherwise, it will be broadcasted to the other MACs except the input port. In the forwarding tasks ingress and egress filtering rules and various statistical information are maintained as well.

III. Layer-2 Switching on GPPs

We have implemented the entire L2 switching task on a LEON2 [5] processor. Software is written in C language and compiled using the gnu compiler gcc ver. 3.2.2. The LEON2 processor is used in a hardware environment shown in Fig. 1. The environment contains four 100Mbps MACs operating in full-duplex mode and one SRAM memory which is used for frame storage.

A. LEON2

The LEON2 processor used in this work is a 32-bit processor based on SPARC V8 architecture. It is a 5-stage pipeline processor designed for embedded applications with on-chip separate instruction and data caches. Communication between external memory and cache controller is managed by the AHB bus. The instruction and data caches are 8 Kbyte 2-way set-associative caches with LRR algorithm.

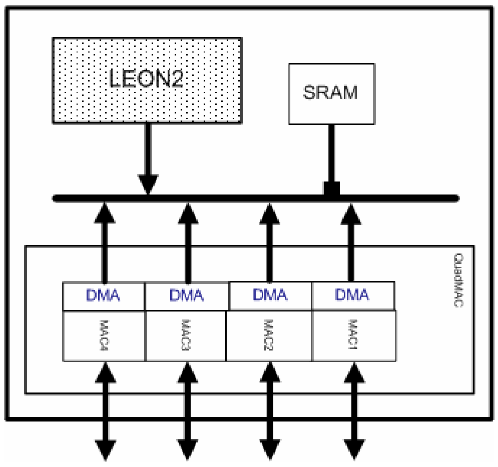

Fig. 1. Processing system based on LEON2 architecture.

IV. Results and Discussion

To evaluate the LEON2 performance in our switching application, we consider the following questions: which workloads do the processor support, what level of performance is required for an L2 switching application, and what type of architecture provides the required performance. The main metric for comparison is the maximum throughput achieved by processor.

A. Throughput

Taking advantage of our instruction-level profiler, and exploiting the simulation results we are able to measure overall system performance in terms of packets per second, as well as lower level performance metrics including Instructions Per Cycle (IPC), and Million Instructions Per Second (MIPS). Our profiling is based on different frame sizes and also different CPU and bus clock frequency ratios. Frames are in the range of minimum frame size (i.e. 64 bytes) to 1024 bytes. CPU and bus clock frequency ratios are 1-1, 2-1, and 3-1 (i.e. CPU clock frequency is three times of the bus clock frequency).

TABLE I shows the number of different instructions used in an L2S flow. As shown in the table, the most frequent instructions are shift and memory instructions. The shift instructions are used in bit manipulation and branch conditions, so a CPU with dedicated bit manipulation instruction could be helpful for this application.

Top

TABLE I. L2S INSTRUCTION COUNT.

Instruction	Occurrence in L2S
load	63
store	43
branch	45
logical	100
shift	142
arithmetic	86
other	148
all instructions	627

As shown in Fig. 1, LEON2 instruction and data memories are both on the AMBA AHB bus so its performance is highly dependent on the instruction and data caches. As shown in TABLE II, the first and second frames are processed in 1293 and 759 clock cycles respectively; the difference is because of the cache misses in the processing of the first frame. To have a fair comparison and conclusion in the following sections we ignore the processing results of the first frame.

TABLE II. L2S INSTRUCTION COUNT FOR 1ST AND 2ND FRAME.

Frames	Instruction count	Clock count	IPC
1st frame	627	1293	0.48
2nd frame	627	759	0.83

The results in TABLE II, are based on the same CPU and bus clock frequencies (i.e CPU frequency or f_{cpu}= bus frequency or f_{bus}). But in our system CPU is at least two or three times faster than bus. So in order to have an actual result we have considered different ratios for CPU and bus frequencies in our simulations. As shown in TABLE III, slower bus and hence more clock counts for L2S flow, reduces the IPC.

TABLE III. L2S CLOCK COUNTS FOR DIFFERENT RATIOS OF F_{CPU} AND F_{BUS}.

Frequency ratios	CPU clock count
$f_{cpu}=f_{bus}$	759
$f_{cpu}=2f_{bus}$	811
$f_{cpu}=3f_{bus}$	857

All of the above results are based on the minimum size frame. TABLE IV shows the L2S instruction and clock count for different frame sizes and different ratios of CPU and bus frequencies (1-2, 1-3 ratios). As shown in TABLE IV, the IPC and MIPS are almost the same for different frame sizes and frequency ratios. But another interesting metric to evaluate the different schemes shown in the table is the system throughput indicated by bits per second (bps). As shown in TABLE IV, when the packet size increases the throughput of the system is increased as well, thus the minimum size of the frame presents the worst case of the throughput in our system.

As shown in TABLE I, about 106 instructions are memory instructions, and 15 of them are from/to non-cacheable memory of the frame. Each bus transaction requires 5 bus clocks and when CPU frequency is three times the bus frequency, each bus transaction requires 15 CPU clock cycles. In the other word, the total bus access to the frame memory for a minimum size frame requires 225 CPU clock cycles (26% of the total execution time). It means that reducing number of bus accesses has a direct impact on the switching performance.

With our simulation infrastructure, we are also able to measure overall system performance, in terms of packets per second. Performance of the L2S software is measured for different frame lengths and TABLE V shows L2S processing times for different frame sizes (CPU frequency =240MHz, bus frequency = 80MHz).

As shown in TABLE V, the worst throughput happens for the minimum size frame. Considering the 672-bit (minimum frame size as shown in TABLE VI), 280Kframe/s can be processed with the 240MHz LEON2 processor. With this processing power almost one 100Mbps (148Kframe/s) line can be processed. In order to support four MACs the required processing time of the minimum frame size should reach 1.7µs (TABLE VI). TABLE V show that the LEON processing power is about 2 times less than the required value.

V. EXTRACTED ARCHITECTURAL GUIDELINES FOR HIGH PERFORMANCE PACKET PROCESSORS

The L2S application basically executes a loop that processes one frame per iteration. The code in each loop is the common path for protocol processing, and it is a few hundred instructions. According to the results, a significant part of the code implements time consuming operations that are required in packet processing, such as bus transactions and bit manipulations.

In order to have a high performance packet processor, we suggest developing some dedicated instructions to reduce bus access time using burst load/store instructions. Each burst operation can be used to read the entire header at once, thus reducing single bus accesses and improving the performance.

Bit manipulation operations can also be reduced to single dedicated instructions. These dedicated instructions can extract various fields of the header and set/reset individual bits and hence, increase processing power.

In order to overcome the memory interlock problem and reduce pipeline stalls for the instructions that are dependent on the load result, compiler considerations are required. In these cases, instruction reordering technique can be employed.

VI. CONCLUSION

The case study investigated in this paper has revealed that in a switching application, some flexibility is provided with GPPs. However, a high performance and yet-flexible packet processor requires some additional custom instructions and dedicated hardware as well. The general-purpose instructions are used to provide more flexibility for future modifications or protocol changes in the application flow and the proposed dedicated instructions are used to boost the performance for repetitive and demanding operations in high speed packet processing.

References

[1] Intel Corporation. December 2001, IXP1200 Network Processor Datasheet, [Online]. Available: http://www.intel.com/design/network/datashts/278298.htm.

[2] Intel Corporation. December 2001. IXP1250 Network Processor Datasheet, [Online]. Available: http://www.intel.com/design/network/datashts/278371.htm.

[3] 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. "IBM PowerNP network processor: Hardware, software, and applications," *IBM Journal of Research and Development*, vol. 47, no. 2/3, pp. 177--194, March/May 2003.

[4] Freescale Semiconductor, Inc. 2005, Motorola C--5 Network Processor Data Sheet, Silicon Revision B0. [Online]. Available: http://www.datasheetcatalog.com/ motorola/17/

[5] Gaisler Research, 2005. [online], Available: http://www.gaisler.com/

TABLE IV. L2S INSTRUCTION COUNT FOR DIFFERENT FRAME SIZES AND DIFFERENT RATIOS OF CPU AND BUS FREQUENCIES.

Frame size	Frequency ratios	Instruction count	Clock count	IPC	MIPS	Throughput (Mbps)
128	$f_{cpu}=2f_{bus}$	627	811	0.77	154	350.38
	$f_{cpu}=3f_{bus}$	627	857	0.73	146	331.6
256	$f_{cpu}=2f_{bus}$	666	857	0.78	155	618.3
	$f_{cpu}=3f_{bus}$	666	900	0.74	148	587.5
512	$f_{cpu}=2f_{bus}$	760	975	0.78	156	1047.6
	$f_{cpu}=3f_{bus}$	760	1028	0.74	148	993.6
1024	$f_{cpu}=2f_{bus}$	963	1268	0.76	152	1580.8
	$f_{cpu}=3f_{bus}$	963	1363	0.71	141	1470.6

TABLE V. TOTAL SWITCHING CAPACITY IN LEON ENVIRONMENT.

Frame size (Bytes)	Switching time (μs)	Throughput (Mbps)
64 (1 BD)	3.57	188.2
128 (1 BD)	3.57	331.6
256 (2 BD)	3.75	587.5
512 (4 BD)	4.28	993.6
1024 (8 BD)	5.68	1470.6

TABLE VI. CALCULATION OF THE REQUIRED SWITCHING TIME.

Frame size= 46B+14(header)+4(CRC)=64B

IFG= 8B(preamble)+12B(inter frame gap)

Total= 64+20=84B=672bit

Frame rate= 148*4=592Kframe/s

Required time for each frame=1.7μs

Top