Performance Evaluation of Queue Management Implementations in Network Processing Units

CH. KACHRIS, TH. ORPHANOUDAKIS, I. PAPAEFSTATHIOU, G. KORNAROS, A. NIKOLOGIANNIS, N. ZERVOS

Ellemedia Technologies
223, Syggrou Av., GR171 21, Athens, Greece,

Abstract: - Network Processing Units (NPUs) target to achieve software programmability like generic CPUs and at the same time achieve the performance of ASICs in order to support wire-speed operation for demanding networking applications. A significant part of the functionality of a NPU is data storage and the implementation of per-flow queuing in order to support the store-and-forward functionality of common network switches and routers. Memory management at high-speeds is not trivial. In this paper we analyze the performance bottlenecks by simulating the actual implementation of a data memory manager integrated in a typical NPU architecture. We expose the performance limitations of software implementations utilizing RISC processing cores typically found in most NPU architectures and identify the requirements for hardware assisted memory management in order to achieve wire-speed operation at gigabit/second rates.

Key-Words: - Network processor, memory management, queue management, DDR-DRAM

1 Introduction

To meet the demand for higher performance, flexibility, and economy in emerging multi-service, broadband networking systems, an alternative to Application Specific Integrated Systems (ASICs), which have been traditionally used to implement packet-processing functions in hardware, the so called Network Processors or Network Processing Units (NPUs), has emerged. NPUs can be broadly defined as System-on-Chip (SoC) architectures that exploit the state-of-the-art in VLSI technology integrating multiple highly optimized processing cores (usually exploiting parallelism and/or pipelining in order to increase throughput). These cores in turn can be characterized either as Reduced Instruction Set Computing (RISC) engines, or dedicated hardware engines for specific complex packet processing functions that require wire-speed performance like classification, per-flow queuing, buffer and traffic management.

Most modern networking technologies (like IP, ATM, MPLS etc.) share the notion of connections or flows (we adopt the term "flow" hereafter) that represent data transactions in specific time spans and between specific end-points in the network for the implementation of networking applications. Depending on the applications and algorithms used, the network processor typically has to manage thousands of flows, implemented as packet queues in the processor packet buffer [1]. Therefore, effective queue management is key to high-performance network processing as well as to reducing development complexity. In this paper we focus on the review of potential implementations within a NPU architecture and performance evaluation of queue management, which is performed extensively in network processing applications.

Our performance evaluation methodology is based on three steps. First we analyze the memory architecture alternatives that can be used for implementing packet buffers and queues of packets in accordance with the overall NPU architecture. Therefore in section 2 we make a brief overview of NPU architectures focusing on the memory management optimizations in each case. In section 3 we analyze the performance bottlenecks examining the accesses to external memories for the implementation of packet buffers in isolation. In section 4 we present an analysis regarding the performance of a queue management implementation on a widely used Network Processor and in section 5 we proceed in a more detailed analysis extending our results for a generic NPU architecture. We perform our measurements on a prototype NPU architecture for a typical implementation of queues of packets with single linked-list data structures and using SRAM and Double Data Rate (DDR) DRAM off-chip memories with the aid of an FPGA development and prototyping platform. The conclusions of our paper are finally summarized in section 6.

2 Memory Management in Network Processors

The requirements with regard to memory management implementations in networking applications stem from the fact that data packets need to be stored in an appropriate queue structure either before or after processing and be selectively forwarded. These queues of packets need to at least serve the First-In-First-Out (FIFO) service discipline, while in many applications flexible access to their data is required (in order to modify, move, delete packets or part of a packet, which resides in a specific position in the queue e.g. head or tail of the queue etc.). In order to efficiently cope with these requirements several solutions based on dedicated hardware have been proposed initially targeting high-speed ATM switching where the fixed ATM cell size favored very efficient queue management ([2], [3], [4]) and later-on extended to management of queues of variable- size packets [5]. The basic advantage of these implementations in hardware is of course the higher throughput with modest implementation cost. On the other hand the functions they can provide (e.g. single vs. double linked lists, operations in the head/tail of the queue, copy operations etc.) needs to be selected carefully at the beginning of the design. Several trade-offs between dedicated hardware and implementations in software have been exposed in [6], in which specific implementations of such queue management schemes in ATM switching applications are examined.

Several commercial NPUs follow a hybrid approach targeting the acceleration of memory management implementations by utilizing specialized hardware units that assist specific memory access operations, without providing a complete queue management implementation. The first generation of the Intel NPU family, the IXP1200, initially provides an enhanced SDRAM unit, which supports single byte, word, and long-word write capabilities using a read-modify-write technique and may reorder SDRAM accesses for best performance (the benefits of this will also be exposed in the following section). The SRAM Unit of the IXP1200 also includes an 8-entry Push/Pop register list for fast queue operations. Although these hardware enhancements improve the performance of typical queue management implementations they cannot keep in pace with the requirements of high-speed networks. Therefore the next generation IXP-2400 provides high-performance queue management hardware that automates adding data to and removing data from queues [7]. Following the same approach the PowerNP ™ NP4GS3 incorporates dedicated hardware acceleration for cell enqueue/dequeue operations in order to manage packet queues [8]. The C-Port/Motorola C-5 NPU also provided memory management acceleration hardware [9], still not adequate though to cope with demanding applications that require frequent access to packet queues, therefore the next-generation Q-5 Traffic Management Coprocessor provided dedicated hardware designed to support traffic management for up to 128K queues at a rate of 2,5 Gbps [10].

3 External DRAM Memory Bottlenecks

A crucial design decision at such high rates is the choice of the buffer memory technology. SRAM provides high throughput but limited capacity, while DRAM offers comparable throughput and significantly higher capacity per unit cost; thus, we chose DRAM. Among DRAM technologies, we focus our analysis on DDR-SDRAM because it performs high performance and it is commonly used, since it is provided at an affordable price.

DDR technology provides 12.8 Gbps peak throughput by using a 64-bit data bus at 100 MHz with double clocking (i.e. 200 Mbps/pin). A DIMM module provides up to 2 GB total capacity and it is organized into 4 or 8 banks in order to provide interleaving (i.e. to allow multiple parallel accesses). However, due to bank-precharging period[1], successive accesses[2] to the same bank may be performed every 160 ns. When a memory transaction tries to access a currently busy bank we say that a bank conflict has occurred. This conflict causes the new transaction to be delayed until the bank becomes available, thus reducing memory utilization. In addition, interleaved read and write accesses also cause loss to memory utilization because they have different access delays[3]. Simulating a behavioral model of a DDR-SDRAM memory and accessing it in a random way, we have estimated the impact of bank conflicts and read-write interleaving to memory utilization. The results of this simulation for a range of banks (1 to 16) are presented in the two left columns of Table 1.

We considered aggregate accesses from 2 write and 2

[1] In this period, a bank is characterized as busy.

[2] A new read/write access to 64-byte data blocks can be inserted to DDR-DRAM every 4-clock-cycles (access cycle = 40 ns).

[3] Write access delay = 40 ns, Read access delay = 60 ns. When write accesses occur after read accesses, the write access must be delayed 1 access cycle.

read ports[4]. By serializing the accesses from the 4 ports in a simple/round-robin order we achieve the throughput loss, presented in Table 1. However, by scheduling the accesses of the 4 ports in a more efficient manner, we can achieve a lower throughput loss by reducing bank conflicts. A simple way to do this is to effectively reorder the accesses of the 4 ports to minimize bank conflicts. It can be performed by organizing pending accesses into 4 FIFOs (1 FIFO per port). Every access cycle the scheduler checks the 4 accesses at the head of FIFOs for conflict and selects an access that addresses a non-busy bank. The information for bank availability is achieved by keeping the memory access history (it remembers the last 3 accesses). In case that more than one accesses are eligible (belong to a non-busy bank), the scheduler selects one of the eligible accesses in round-robin order. In case that no pending access is eligible, then the scheduler sends a no-operation to the memory, losing an access cycle. The results of this optimization are presented in Table 1. Assuming organization of 8 banks, the optimized scheme reduces throughput loss by 50% of the not-optimized scheme.

Table 1: DDR-DRAM throughput loss using 1 to 16 banks

banks	No Optimization		Optimization	
	Throughput Loss		Throughput Loss	
	Bank conflicts	Bank conflicts + write-read interleaving	Bank conflicts	Bank conflicts + write-read interleaving
1	0.750	0.75	0.750	0.750
2	0.647	0.66	0.552	0.660
3	0.577	0.598	0.390	0.432
4	0.522	0.5	0.260	0.331
5	0.478	0.48	0.170	0.290
6	0.442	0.46	0.100	0.243
7	0.410	0.42	0.080	0.220
8	0.384	0.39	0.046	0.199
9	0.360	0.376	0.032	0.185
10	0.338	0.367	0.022	0.172
11	0.321	0.353	0.018	0.165
12	0.305	0.347	0.012	0.159
13	0.289	0.335	0.010	0.153
14	0.275	0.33	0.007	0.148
15	0.264	0.32	0.004	0.143
16	0.253	0.317	0.003	0.139

From the above we conclude that only a percentage of the nominal 12.8 Gbps peak throughput of a 64-bit/100 MHz DDR-DRAM can be utilized and the design of the memory controller must be an integral part of the memory management solution.

4 Evaluation of a Queue Management on the IXP1200 Network Processor

As it was described in Section 2, the most straightforward implementation of memory management in NPUs is based on software executed by one or more on-chip microprocessors. Apart from the memory bandwidth that we examined in isolation in the previous section, a significant factor that affects the overall performance of a queue management implementation is the combination of the processing and communication latency[5] of the queue handling engine (either generic processor or fixed/configurable hardware) and the memory response latency. Therefore the overall actual performance can only be evaluated at system level. Since Intel's IXP1200 is one of the most widely used NPUs today and represents a typical NPU architecture in this section we provide results regarding the maximum throughput that can be achieved when implementing memory management in IXP1200 software.

The IXP1200 consists of 6 simple RISC processing microengines [7] running at 200MHz. When porting the queue management software to those RISC-engines, special care should be given so as to take advantage of the local cache memory (called "Scratch memory") as much as possible. This is because any accesses to the external memories take a very large number of clock cycles. One can argue that using the multithreading capability of the IXP can hide this memory latency. However, as it was proved in [11], the overhead for the context switch, in the case of multithreading , exceeds the memory latency and thus this IXP feature cannot increase the performance of the memory management system, when external memories should be accessed.

Even by using a very small number of queues (i.e. less than 16), so as to be able to keep every piece of information in the local cache and in the IXP's registers, each Microengine cannot service more than 1 Million Packets per Second (Mpps). Or, in other words, the whole of the IXP cannot process more than 6Mpps. Moreover, if 128 queues are needed, and thus some external memory accesses are necessary, each microengine can process at most 400Kpps. Finally, for 1K queues the peak bandwidth that can

[4] A write and a read port from/to the network, a write and a read port from/to the internal processor.

[5] Communication with the peripheral memories and memory controllers

be serviced by all 6 IXP microengines is about 300Kpps [12]. We summarize the above throughput results in the following table.

Table 2: Maximum Rate Serviced when queue management runs on IXP 1200

Num of Queues	1 Microengine	6 Microngines
16	956 Kpps	5.6 Mpps
128	390 Kpps	2.3 Mpps
1024	60 Kpps	0.3 Mpps

From the above it can easily be derived that this software approach, cannot cope with today's state-of-the-art network links, if the network application involves the handling of more than a hundred separate queues.

5 Evaluation of a Custom Software Implementation of Queue Management

In order to experiment with different design alternatives and perform detailed measurements of a typical queue management implementation with the aid of the embedded processing cores of an NPU we implemented the basic hardware set-up of a typical NPU. With the aid of state-of-the-art FPGAs that provide hard macros of embedded RISC cores as well as synthesizable RISC modules and the respective development tools we implemented the core design of an NPU. The architecture of the system, targeting the Xilinx Virtex-II Pro Platform, is depicted in Figure 1. As it is shown, the 64-bit Processor local bus (PLB) is used as the system bus at a clock frequency of 100MHz. The PowerPC 405 is a 32-bit implementation of the IBM PowerPC ™ embedded environment architecture and is used as the central processor. The OCM Controller is used to connect the PowerPC with the Instruction and the Data Memory (16KBytes each). The size of the code used for memory management is small enough to fit in the instruction memory. The packet buffer is implemented in an external DDR DRAM -as justified above- using a standard PLB DDR controller, while the queue information (mainly pointers) is stored in an external ZBT SRAM, using the PLM External Memory Controller (EMC). To simulate real network traffic an Ethernet MAC port has been used. The MAC Core (provided by OpenCores) core uses two WishBone ports. The first port is attached to the PLB Bus, through the PLB to WishBone Bridge, and is used for control. The second port is attached to a Dual Port internal Block RAM (4 Kbytes), and is used to store temporarily the in-coming and out-going Ethernet packets. With the aid of this on-chip dual-port RAM (DP-BRAM) data transfers between the network interface and the queue manager (i.e. processor and buffer memory) can be achieved very efficiently.

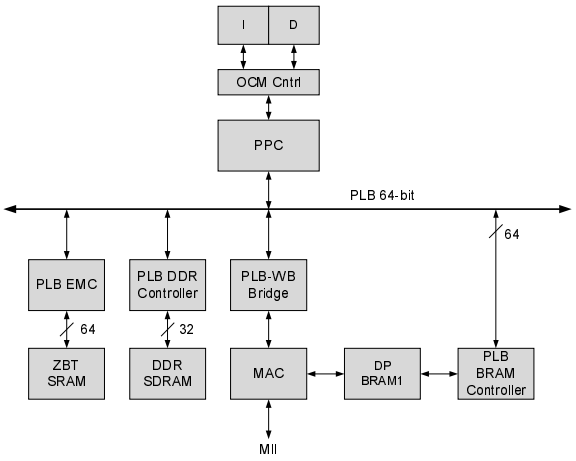

Figure 1: NPU core architecture set-up on the Xilinx Virtex-II Pro PLD platform with off-the-self components (IP cores)

5.1. Configuration

The PowerPC has been configured to use the instruction and data cache, both in write back mode. The PowerPC and PLB Bus clock frequency has been set to 100MHz and the PLB DDR controller is configured in burst mode. Finally, the code has been compiled using GCC optimization level 2. The frequency selection was dictated by the FPGA timing requirements so we chose to operate the CPU at the speed of the PLB bus and the memory controller, which in turn could not reach the maximum CPU frequency. A state of the art embedded RISC core though (like the PowerPC core provided in the Xilinx Virtex II Pro family) can reach operation frequencies in the range 200-300 MHz, so we also compare the performance in the projected range of frequencies (that could be achieved if the design of Figure 1 is implemented in custom VLSI technology). Note that the design of Figure 1 represents a typical organization of an NPU core design, where the PowerPC is used as a typical on-chip micro-engine[6] (and even more powerful when compared to an IXP1200 micro-engine).

[6] One of the potentially many on-chip micro-engines, since it is typical to assign one micro-engine to implement queue management operations [7]

5.2. Queue structure

We implemented a typical Data Memory Manager (DMM) handling queues of packets implemented as single linked-lists. The incoming data items are partitioned into fixed size segments of 64 bytes each. Then, the segmented packets are stored in the data memory, which is segment aligned. Our DMM implementation organizes the incoming packets into queues and handles and updates the data structures kept in the pointer memory. Each segment has a pointer that targets the next segment. A free list is used to allocate new segments and a table of queues is used for storing the segments.

The following functions have been used to control the Queue Manager:

- Enqueue Segment
- Dequeue Segment
- Enqueue Free List
- Dequeue Free List

The packet operations are analyzed into segment and free list operations. For example, the enqueue packet operation is analyzed into the following steps: First a new pointer is allocated from the free list, then this pointer is stored to the queue list and then the data are transferred to the memory.

5.3. Performance evaluation.

The Table below shows the number of cycles for the execution of each operation.

Table 3: Cycles per packet operation (DRAM controller support for burst transactions disabled)

Enqueue Packet	
Function	**Cycles**
Dequeue Free List	34
Enqueue Segment	46/68*
Copy a segment	136
Total	*216/238*

*46 cycles for the first segment of the packet, 68 for the following segments

Dequeue Packet	
Function	**Cycles**
Dequeue Segment	42
Enqueue Free List	52
Copy a segment	136
Total	*230*

For a 100-Mbit/s network and a minimum packet length of 64 bytes the available time to serve this packet is 5.12 μsec. The PowerPC's clock frequency is set to 100 MHz, thus the available headroom is 512 clock cycles to serve each packet for a half-duplex network, or 256 cycles for a full duplex network. This means that all the available processing capacity of the PowerPC core has been used for the queue management, and it cannot afford to further process the packet, thus another processor must be used for further processing. The majority of the cycles are wasted in memory latency and transactions over the PLB bus. Even if the processor operation frequency is set to 400MHz the improvement of performance would be negligible, since the maximum frequency of the PLB bus in reconfigurable logic is 100MHz.

As shown in Table 3, half of the cycles are used to copy the data of the segment. A major improvement is to exploit the line transactions of the PLB. The PowerPC can execute line transactions over the bus using the data cache unit [13]. Using this configuration a segment can be retrieved from the BRAM and stored into the data cache in only 12 cycles (9 cycles for 9 double words and 3 cycle latency). Thus, the total number of cycles to copy a segment becomes:

$$T_C = (T_R + T_l) + (T_W + T_l) = 2*(9+3)=24 \text{ cycles}$$

,where T_R denotes the number of cycles to read a segment from the on-chip buffer (Xilinx BRAM block), T_W denotes the number of cycles to write a segment to the DDR DRAM and T_l denotes the 3-cycles bus latency. Thus, the total number of cycles to enqueue and dequeue a packet becomes 128 and 118 respectively, which dictates that the PowerPC would sustain up to about 200 Mbits/sec throughput.

Another improvement would be the use of a DMA controller [14]. In this case, four 32-bit registers have to be set before each transaction [15];

- the DMA control register,
- the source address register,
- the destination address register and
- the length register.

Each single PLB write transactions needs 4 cycles to execute, thus we need at least 16 cycles to initiate the DMA transfer and at least 34 cycles to copy the data from BRAM to DRAM or vice versa. Note that the total time per operation is approximately the same as before. Hence, the overall throughput does not increase significantly but in this configuration the processor has more headroom for other processes (due to the offloading of data copying tasks to the DMA engine).

6 Conclusions

Due to the store and forward architecture of packet forwarding systems and the complexity of protocol processing and traffic management functions in high-

speed networking applications memory management and the handling of packet queues is a major bottleneck and has been an application field of state-of-the-art NPUs. We have analyzed the memory access overheads and the processing requirements of queue manipulation functions. Our performance evaluation of queue management implementations both on the commercial IXP1200 NPU platform as well as on our custom prototype architecture exposes the significant processing resources required when general-purpose RISC engines are used to implement queue manipulation functions. Even with state-of-the-art VLSI technology and processor frequencies in the order of several hundreds MHz a single processor can only achieve a throughput in the order of hundreds of Mbps for a moderate number of queues. Hence, processing power by itself does not suffice to support queuing in Gbit/sec links but careful selection and control of the external DRAM as well as specialized hardware to accelerate queue management operations is an integral part of the NPU design exploration space that needs to be optimized in order to maximize the overall system performance at an acceptable cost.

References:

[1] V. Kumar, T. Lakshman, and D. Stiliadis, "Beyond best-effort: Router architectures for the differentiated services of tomorrow's internet," IEEE Communications Magazine, pp. 152-164, May 1998.

[2] J. Rexford, F. Bonomi, A. Greenberg, A. Wong, "Scalable Architectures for Integrated Traffic Shaping and Link Scheduling in High-Speed ATM Switches," IEEE Journal on Selected Areas in Communications, pp 938-950, June 1997.

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

[4] D. Whelihan and H. Schmit, "Memory optimization in single chip network switch fabric", Proceedings of the 39th conference on Design automation, New Orleans, Louisiana, USA, June 2002.

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

[6] D. N. Serpanos, P. Karakonstantis, "Efficient Memory Management for High-Speed ATM Systems", Design Automation for Embedded Systems, , Kluwer Academic Publishers, 6, 207–235, April 2001.

[7] S. Lakshmanamurthy, et. al. "Network Processor Performance Analysis Methodology", Intel Technology Journal Vol. 6 Issue 3, 2002.

[8] James Allen, et. al. "PowerNP network processor: Hardware, software and applications," IBM Journal of Research and Development, vol. 47, no. 2-3, pp. 177-194, 2003.

[9] C-port Corp., "C-5 Network Processor Architecture Guide", C5NPD0-AG/D, May 2001.

[10] Motorola Inc., "Q-5 Traffic Management Coprocessor Product Brief", Q5TMC-PB, December 2003

[11] W. Zhou, C. Lin, Y. Li, Z. Tan, "Queue Management for QoS Provision Build on Network Processor" 9th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'03) May 28 - 30, 2003 San Juan, Puerto Rico

[12] Tammo Spalink, Scott Karlin, Larry Peterson, Yitzchak Gottlieb, "Building a Robust Software-Based Router Using Network Processors", 18th ACM Symposium on Operating Systems Principles (SOSP'01), Chateau Lake Louise, Banff, Alberta, Canada, October 2001.

[13] PowerPC 405 Processor Block Reference Guide, September 2, 2003, Xilinx Inc.

[14] Meleis, H.E., Serpanos, D.N., "Designing communication subsystems for high-speed networks", IEEE Network, Vol. 6, Issue 4.

[15] Direct Memory Access and Scatter Gather Datasheet, Version 2.2, January 2003, Xilinx Inc.

Acknowledgments

This work was performed in the framework of the Eureka E! 3326-WEBSoC project, which is partially funded by the Greek Secretariat of Research & Technology. The authors would like to thank Dr. Nikos Nikolaou for providing valuable help to the implementation of the queue manager in software.