Performance Analysis by HDL Simulation of Network Interface Design Alternatives

Waseem M. Haider, Julio Ortega, Antonio Francisco Díaz

***Abstract*— As the bandwidth of network links has increased at an exponential rate to accommodate the application requirements, efficient network interfaces has become decisive. Taking into account the trend towards multi-core microprocessors and programmable processors in the NIC, a plausible goal is to distribute the communication overhead amongst all the processing units available in the computer, not only to improve the communication throughput and latency, but also to leave more CPU cycles available for applications and operating system tasks.**

This paper deals with the evaluation by simulation of the HDL (Hardware Description Language) models that implement different network interface design alternatives (offloading and onloading approaches) to improve the network interface performance by taking advantage of the different processing cores available in the server node.

By using an HDL model of the communication path we have developed, we provide a fair comparison of onloading and offloading alternatives, and analyzes the obtained results with the aid of the previously proposed LAWS model. This way, we contribute to understand the effect of the different elements in the nodes that take part in the communication path, particularly, we consider the technology of the processors in the node, the characteristics of the main and cache memories, the bandwidth of the I/O buses, and the profiles of the communication/computation rates of the applications. The experimental results show the accuracy and usefulness of our simulation model and provide an approximate validation of the LAWS model.

***Index Terms*—Hardware Description Language, LAWS model, protocol offloading, protocol onloading.**

I. INTRODUCTION

The improvements in the computer applications increasingly depend on the communication technology. The bandwidth of the Ethernet links has increased at exponential rate to accommodate the requirements of the present applications. Today, 1 Gb/s Ethernet is usual for desktop systems, and 10 Gb/s Ethernet links are available for server systems. A full-duplex 10 Gb/s link can deliver maximum sized 1,518 byte frames at the rate of 812,744 frames per second in each direction. Therefore, a full-duplex 10 Gigabit Ethernet controller must be able to sustain about 435 MIPS [1] and 4.8 Gb/s of data bandwidth for protocol processing. So, we should need processors between 100 and 200 GIPS to cope with these bandwidths [1]. The communication path processing includes the I/O bus control, memory latencies management, interrupts, and other overheads that do not scale with faster processors [20]. The implementation of a suitable network interface should take into account all those parameters and operations that do not scale properly.

Over the last few years, the researching work on network interfaces has been focused on reducing the time overhead of the communication system architecture by using light-weight communication protocols like GAMMA [2] or CLIC [21], or through user-level network interface protocols [3]. Nevertheless, the bandwidth increases in the network links have made it necessary to consider new strategies to improve hardware elements, such as the increasing in the bandwidth of the I/O buses and in the processing capabilities of the network interface cards. On the other hand, there are important practical problems related to light-weights protocols. First of all, there is no compatibility with the existing applications that use sockets and, in some cases, with the infrastructure based on IP [4]. Among the solutions proposed to cope with these problems is the developing of high performance sockets at user level [4], the modification of the socket interface to maintain compatibility with IP [5]. In [7], it has been shown that it is possible to reach adequate performance levels in I/O intensive applications without accessing to the NIC at user level.

All the NICs for 1 Gbps and 10 Gbps Ethernets are able to determine and check the TCP/IP checksum. Generally, there are another ways to reduce the interrupt frequency by generating one interrupt for a large number of packets [3]. Jumbo frames (frames up to 9000 bytes), used to reduce the pre-frame processing overhead, and header splitting, which is used to place protocol headers and data in separated buffers [22]. Beside all the features that mentioned above, many NICs include programmable processors.

An alternative that has been also proposed is protocol offloading [6]. The idea behind this alternative is the distribution of the communication tasks among the different elements in the host, mainly among the host CPU and other processors in the NIC. It implies to release the communication load from the central processor to another processor. The tasks that imply interaction with the network is implemented in the NIC in order to leave more CPU cycles for the computation required by the applications. In this way, protocol offloading can be seen as a technique that enables

Manuscript received December 3, 2009. This work was supported in part by the Department of Computers Architecture and Technology under TIC-1359 project (Junta de Andalucía, Spain).

Waseem M. Haider. Author is with the Department of Computers Architecture and Technology, University of Granada, Granada, CO 18071 Spain (phone: +34-695-856383; fax: +34-958-248993; e-mail: mhaider@atc.ugr.es).

Julio Ortega. Author is with the Department of Computers Architecture and Technology, University of Granada, Granada, CO 18071 Spain (phone: +34-958-243228; e-mail: julio@atc.ugr.es).

Antonio Francisco Díaz. Author is with the Department of Computers Architecture and Technology, University of Granada, Granada, CO 18071 Spain (phone: +34-958-246127; e-mail: afdiaz@atc.ugr.es).

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

the parallelization of the network communication tasks and the direct data placement on the main memory, thus avoiding some communication overheads rather than only shifting them to the NIC [8].

The use of processors in the NIC to accelerate the processing of communication tasks (offloading) has been proposed in numerous works, that present diverse proposals to optimize and distribute the communication tasks between the CPU and the processor of the NIC [8], [9]. There are also many commercial proposals of TOE (TCP/IP Offloading Engines), the hardware that helps offloading to be implemented [10]. Besides releasing the CPU from part of the work related to communication, protocol offloading to the NIC can provide other advantages. The first one is to reduce the message latency because, as protocols are implemented in the NIC, short messages like the ACKs do not have to cross the I/O bus. This also supposes a reduction in the number of interruptions that the CPU has to process to attend the received messages. Other advantages are the improvement in the efficiency of DMA transfers from NIC corresponding to short messages and the possibility to dynamically manage the protocols in the NIC in order to select the most suitable one to build the messages according to the communication protocol information and the destination. Nevertheless, certain controversy exists about the efficiency of the offloading approach. More specifically, in [11]-[13], some objections are provided about the utility of this technique taking into account that the processor in the NIC usually presents worse processing performance than the CPU. Other reasons argued for the uncertainty on the offloading utility refer to the difficulties that arise for its implementation, test, and maintenance.

Other alternative, called onloading, considers the use of some of the available cores in a multicore microprocessor or CPUs in an SMP [13]-[15]. This trend agrees with the mainstream tendency in the development of multi-core architectures. There are commercial proposals of onloading as the I/OAT (Intel I/O Acceleration Technology) [16] technique. With onloading, one of the cores executes the communication software, while the rest of the software (applications and the rest of the operating system) is executed in other cores. Thus, the processing of the network interface is done in a core with the same characteristics and privileges in the accesses to main memory as the cores where the application is executed. On the other hand, it is possible to send all the interruptions generated by the NIC towards the core in charge of the network interface processing. In the case of the offloading technique, the central processor must still attend to some interruptions generated by the NIC.

It is difficult to analyze the efficiency of offloading and onloading to optimize the network interface due to the interaction among the different hardware elements, the operating system, and the application tasks. With the integration of multiple processor cores on the same chip, more and more sophisticated caches are appearing on the scene while the size are growing at the same time. We will show that the network protocol software is very sensitive to the cache behavior, thus being relevant to communication throughput performance under different conditions such as which is shown in [23]. In any case, it is difficult to find systems with similar hardware and software characteristics to make a fair comparison between the offloading and onloading alternatives. Thus, in this paper we have used a HDL (Hardware Description Language) simulation model of the communication path to compare the performance of both alternatives under the influence of the different elements that characterize the network interface.

The rest of the paper is organized as follows. After this introduction, Section 2 gives an overview of the LAWS model. This model helps us to organize the parameters of the simulation model in order to make easier the search in the offloading and onloading design spaces and which has been proposed to predict oflloading/onloading effects. In section 3, we present our HDL simulation model of the communication path, which is used to analyze the behavior of the different elements in the communication path. Finally, Section 4 and 5 provide the experimental results and the conclusions, respectively.

II. PREDICTION PERFORMANCE OF OFFLOADING /ONLOADING BY USING LAWS MODEL

Some performance models have been proposed to understand the fundamental principles that are on the basis of the experimental results obtained after applying some strategies to optimize the network interfaces. In [17], a theoretical model called LAWS is presented. It tries to characterize the improvement provided by protocol offloading in the NIC, for applications related to Internet services or streaming applications. Moreover, we have also used LAWS to predict the performance of the onloading alternative because this model can in fact be applied whenever the communication overhead are distributed between the different processors in the server node. Thus, it is possible to compare both alternatives.

The LAWS model is based on the estimation of the peak throughput of the pipelined communication path according to the throughput provided by the correspondent bottleneck (the network link, the NIC, or the host CPU). A LAWS only considers applications that are throughput limited and fully pipelined. The analysis provided in [17] considers that the performance of CPU is limited before applying the protocol offload technique. In the following, we provide a brief description of the LAWS model for completeness.

Figure 1, explains how the LAWS model views the system with and without offloading. We have used the same notation than in [17] to derive the peak throughput functions. Before offloading, the system is considered a pipeline with two stages, the host and the network. In the host, to transfer m bits, the application causes a CPU work equal to aXm and the communication produces a CPU work equal to oXm. Where a and o are the amount of the CPU work per data unit, and X is a scaling parameter used to take into account variations in the processing power with respect to a reference host. Moreover, the network link latency to provide m bits when the bandwidth of the link equals B, is m/B. Therefore, the peak throughput provided before offloading is determined by the bottleneck stage B_{before}=min (B, 1/(aX+oX)). After offloading, we have a pipeline with three stages, and a

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

portion p of the communication overhead has been transferred to the NIC. In this way, the latency stages for transferring m bits are m/B for the network link, $aXm+(1-p)oXm$ for the CPU stage, and $poY\beta m$ for the NIC stage. In the expression of the NIC latency, Y is a scaling parameter to take into account the difference in processing power with respect to a reference, and β is a parameter that quantifies the improvement in the communication overhead that could be reached with offloading. This way, βo is the portion of the normalized overhead that remains in the system after offloading with p=1 (complete offloading). Thus, after offloading, the peak throughput is B_{after}=min (B, $1/(aX+(1-p)oX, 1/poY\beta m)$ and the total improvement in peak throughput is defined as $(B_{after}- B_{before})/B_{before}$.

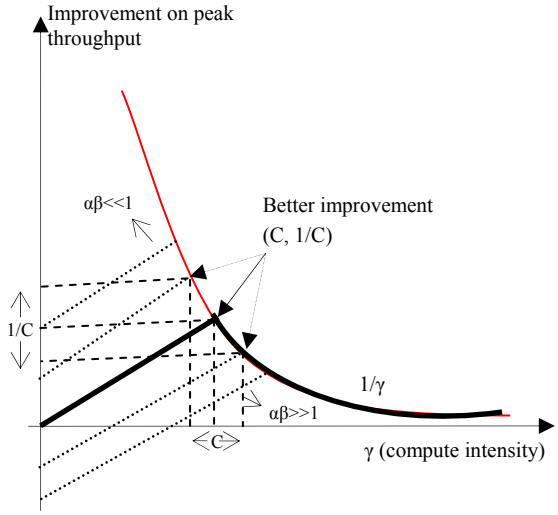

Fig.1 Behavior of the peak throughput improvement according to LAWS model.

The LAWS acronym comes from the parameters used to characterize the offloading benefits. Thus, along with the parameter β (**S**tructural ratio), we have the parameter $\alpha=Y/X$ (**L**ag ratio) which considers the ratio between the CPU speed and the NIC computing speed. The parameter $\gamma=a/o$ (**A**pplication ratio) measures the computation/communication ratio of an application and the parameter $\sigma=1/oXB$ (**W**ire ratio) corresponds to the portion of the network bandwidth that the host can provide before offloading. According to the parameters of the LAWS model (α, β, γ, and σ), the relative improvement of the maximum throughput comes from the following expression:

$$BW_{before}=\min(B, 1/(aX+oX)) = \min[1/\sigma, 1/(\gamma+1)] \quad (1)$$

$$BW_{after}=\min(B, 1/(aX+(1-p)oX), 1/poY\beta) = \min[1/\sigma, 1/(\gamma+(1-p)), 1/p\beta\alpha] \quad (2)$$

$$\delta b=[\min(1/\sigma, 1/(\gamma+(1-p)), 1/p\beta\alpha) - \min(1/\sigma, 1/(\gamma+1))] / [\min(1/\sigma, 1/(\gamma+1))] \quad (3)$$

From the expression (3) and Figure 1, some conclusions can be obtained:

1. Protocol offloading is useful in applications with computation/communication rate, γ, very low. This profile corresponds to stream data processing application and network storage serves with a large number of disks, etc. For CPU intensive applications, the throughput improvement reached by offloading is bounded by $1/\gamma$ and tends to zero as the computation cost increases (i.e. γ grows). The maximum improvement that can be reached is obtained by γ =max ($\alpha\beta$, σ). Also, the improvement slope (γ+1)/c-1 is 1/c being c=max ($\alpha\beta$, σ), the bandwidth improvement grows more while the $\alpha\beta$ and σ decreases.

2. When (α<1), the NIC speed is lower than the CPU speed, offloading may reduce the network interface performance. If the NIC gets saturated before the network link ($\alpha\beta<\sigma$), the improvement is bounded by 1/α. Nevertheless, if an efficient offload implementation (for example Direct Data Placement [11]) allows structural improvements (a reduction in β), it is possible to maintain the offloading usefulness for α>1.

3. When (σ>>1), the host is able to assume the communication overhead without aid, and there is not any improvement with offloading/onloading in slow networks. The offloading usefulness can be high when the host is not able to communicate at link speed (σ<<1), but in these circumstances, γ has to be low, as it has been previously mentioned. Thus, the offloading can be seen as a very useful technique as there is a trend to a faster network (σ decreases).

4. When σ is near one, the best improvement corresponds to those cases that present some balance between computation and communication before offloading ($\gamma=\sigma$=1).

However, it is clear that the system communication performance depends on many parameters; from the application (computation/communication) profile to the interaction between operating system, application, and hardware. As it has been said before, although the LAWS model in [17] is applied to offloading, it is also possible to use this model when the execution of the communication tasks is done in a processor located in other parts of the system like SMP or CMP. Therefore, we will use the LAWS model as a base to get insight to the simulation results for offloading and onloading.

III. AN HDL SIMULATION MODEL FOR THE COMMUNICATION PATH

Simulation is a suitable approach to evaluate different alternatives in computer architecture design. An useful simulator needs to model the machine with the details required by the problems to be solved. There are many simulators with characteristics required by the network interface analysis like M5 [24], SIMICS [26] and SimOS [25].

In this paper we have used an HDL simulator through an HDL model we have developed. HDL models allow us to have an idea of the implementation of the new proposals, to complement the results obtained by functional simulators, and to generate estimations of the clock speed that can be reached by the hardware. They also make possible to identify

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

the critical paths and to improve their design aspects at a level nearest to the hardware details [18].

Firstly, in this section we describe our HDL simulation model in detail. It has been developed to study the conditions in which the offloading and onloading alternatives may contribute to the bandwidth improvement results. As it has been mentioned, this model allows us the simulation of the hardware characteristics at an adequate level to understand the influence of the different implementation alternatives. Our HDL simulation model also makes it possible the simulation of codes and their interaction with hardware and operating system. This interaction has been modeled by means of delay magnitudes that produce different overhead sizes associated with the processing of protocol and drivers, etc. It is difficult to generate the LAWS curves of the throughput improvement against the application ratio, when one of the other parameters changes while the others are held constant (Figure 1). However, our HDL simulation model makes this goal easier to reach.

The different modules of our HDL simulation model for the communication path (including the different alternates: i.e. base system, offloading and onloading) are shown in Figure 2.

Besides the NIC, the CPU the cache, the chipset, and the main memory, we have also included the delays in the I/O and memory buses. It is possible to inject packets of different sizes, from the memory of the sender by using two alternatives: a packet generator or a trace file.

The communication path in our simulation model contains the memory module *Memory* in the sender part. Packets with different sizes and speeds are generated from this module. These packets reach to the *NIC* module, where they are stored in a queue of buffers. Then, the *Network* module reads packets from this queue. The simulation model also includes various modules of *CPUn* that correspond to different CPU present in the host, and the *chipset* module used to implement the interface between the different processors and the *I/O* bus module.

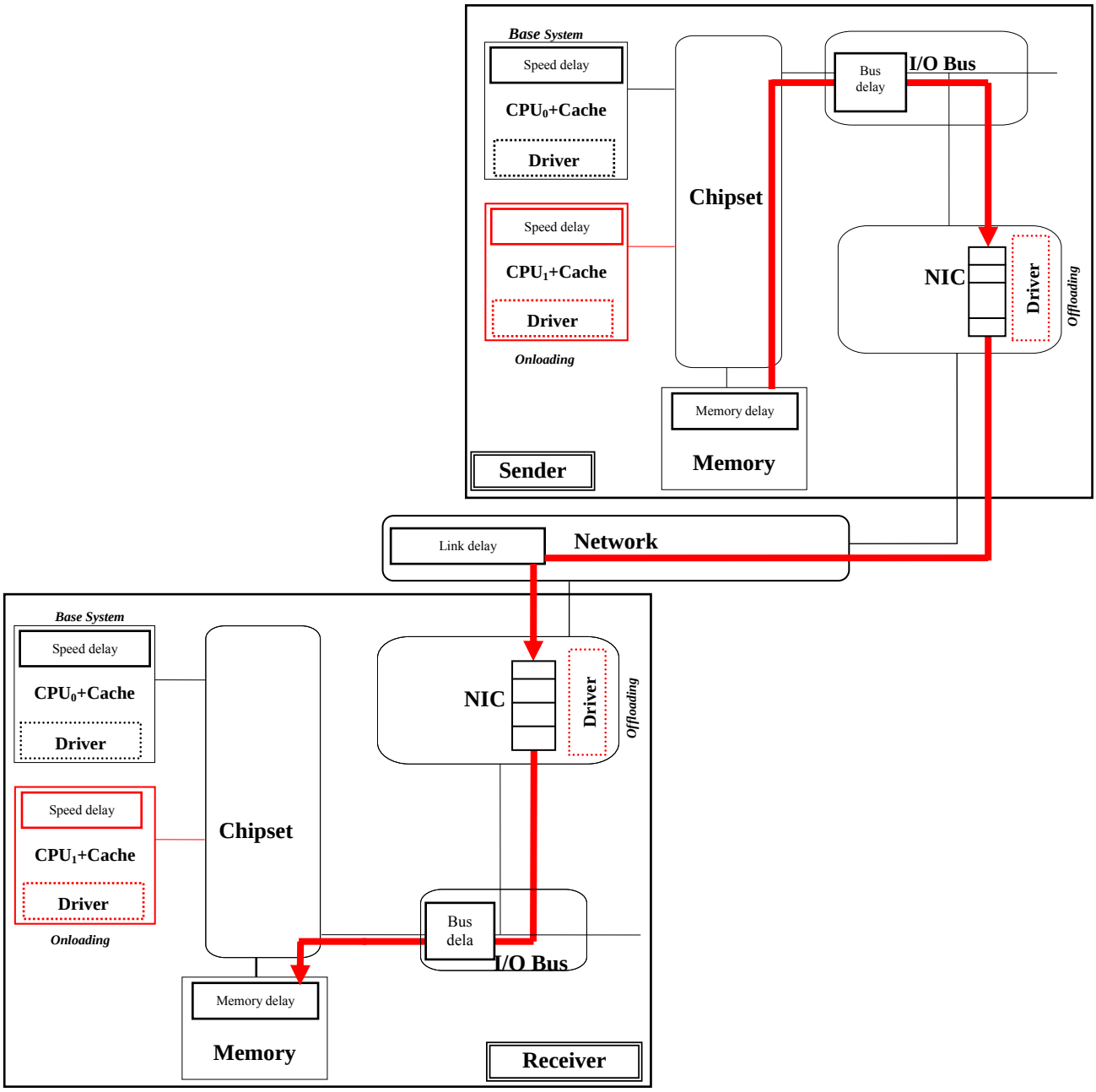

Fig. 2 Modules of the HDL simulation model with alternatives (Base system, Onloading and Offloading).

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Figure 3 shows all the elements in the *NIC* module, including buffers to store the data coming from the network link module and from the I/O bus.

To read or write from the NIC, there are two modules to control the data path. The main point in the NIC module is to implement the communication protocol offloaded by using the driver (module Driver). The other tasks implemented in the NIC are: the DMA initialization by using DMA signal and the interrupt request to the CPU by using Interrupt generation. Other control signals (*prodReady*, *consReady*, *cReady_interface*, and *pReady_interface*) are handshaking signals used to control the read/write transfer. The CPU and the processor in the NIC interact by reading from or writing to some shared registers.

Figure 4 illustrates the main elements of the *CPU* module. It shows the two main parts for processing (*Applications and Communication* boxes in Figure 4), the cache memory, and the interrupt controller (that use *Interruptack* and *Interrupt* signals to manage the data processing with network interface card). In the base case (implementing neither offloading nor onloading), all the application and communication tasks are executed in the CPU module, while in the case of offloading and onloading, the communication tasks will be moved to the *NIC* or to the CPU_1 modules, respectively.

The NIC and CPU modules can be simulated according to different alternatives depending on the technique used to optimize the network interface. The NIC module behavior is controlled by a program stored in the memory of the NIC.

It executes the protocol when offloading is implemented. In this case, the CPU avoids almost all the communication overhead, as the NIC is able to process the communication protocol.

In the base case, when the information from both directions (send or receive) reaches to the NIC, it interrupts directly the CPU. The CPU executes the driver to initialize the DMA operation between the NIC and the main memory. The NIC stores the received data in the main memory through the DMA operation and informs the CPU at the end of the operation. Then, the CPU starts the process of the packet stored in the main memory. In the offloading case, the NIC processes all the packets, and then it interrupts the CPU that executes the driver to initialize the DMA operation as in the base case. After that, the NIC starts the DMA operation to transfer packets to the main memory and informs the CPU that the data is available in the main memory. We have also implemented another way to release the CPU from the communication overhead when the NIC is able to process the protocol and to initialize the DMA. When these packets are stored in the corresponding addresses, the NIC informs the CPU that the application can use them. In the offloading case, only one CPU module is simulated and in the onloading case, two CPU modules are used as shown in figure 2. In the first alternative, all the applications and the rest of the tasks of the operating system are executed in the CPU_0, and in the second one, all the tasks related to the communication processing are executed in the CPU_1.

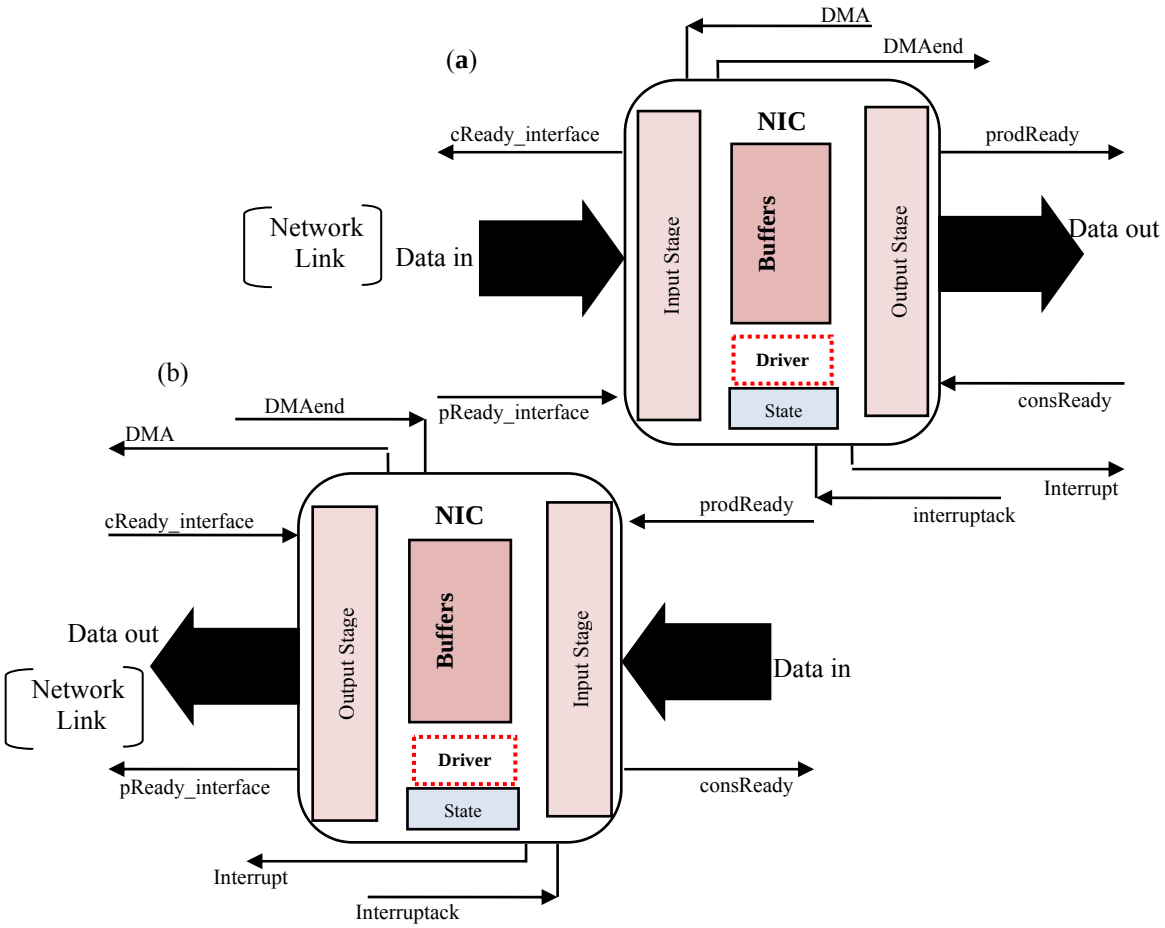

Fig.3 Elements of the Network Interface Card module NIC in the ((a) Receiver and (b) Sender) parts.

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

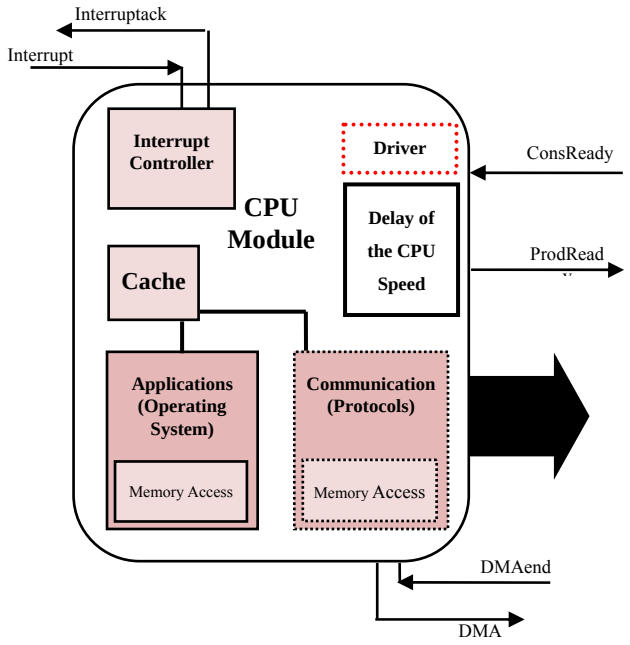

Fig.4 Elements of the CPU module in HDL simulation.

Figure 5 shows the experimental signals among the different modules in the HDL simulation corresponding to the different alternatives. The interrupt signals are exchanged between the CPU and NIC modules. In Figure 5 the signal *Interrupt* is used for interrupt requesting and the signal *Interruptack* is used for the acknowledgment. The DMA signal indicates a direct memory access transfer from the NIC to the CPU.

The protocol signal indicates that the protocol is processed by CPU module (in the base system), while Driver signal indicates that the protocol processed in the NIC (offloading), in the last case (onloading). The CPU1 module is used to process the protocol of the communication.

In the receiver side, the packets enter to the NIC module and stored in the NIC buffers. In the base system as shown in figure (5. a), (1) the CPU is interrupted by the signal Interrupt, and it initializes the DMA operation and transfers the incoming data to the Memory module by using the DMA signal while the operation ends by using DMAend signal. After that, (2) the CPU module processes the packet by using the protocol signal. In the case of offloading, as shown in figure (5. b), (1) the packets are firstly processed in the NIC module implemented through Driver, and (2) the CPU module is interrupted to start the transfer of the data. In the case of onloading, as shown in figure (5. c), the CPU_1 modules are used as processor of the NIC to process the communication protocols while protocol signal is active (2).

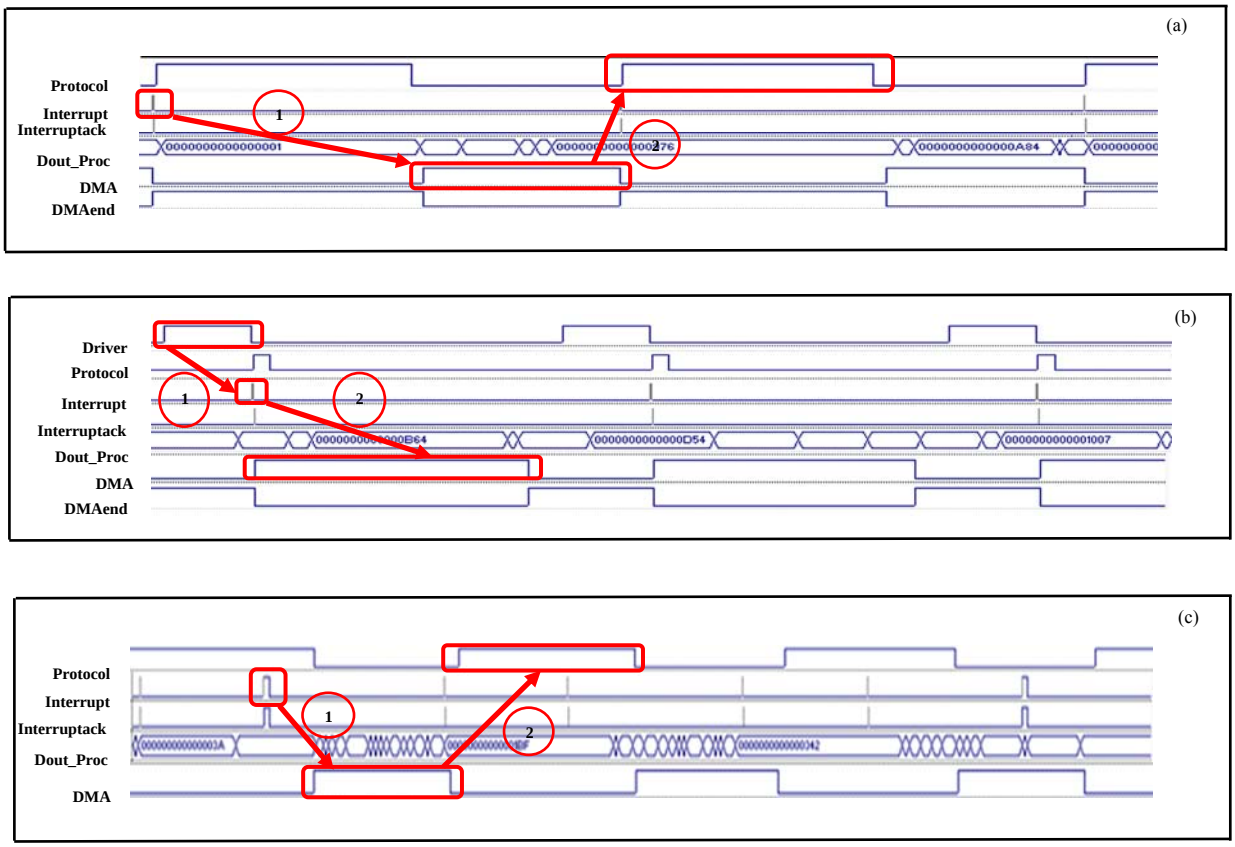

Fig.5 Timing diagrams for three alternatives (a. base system, b. offloading, c. onloading).

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

IV. EXPERIMENTAL RESULTS

In all our experiments, we have compared the behavior of the offloading and onloading strategies with respect to the LAWS theoretical model, as this model helped us to organize the space of the offloading and onloading strategies. The LAWS model provides a way to understand the performance under different application. In our HDL simulations, we have supposed that the offloading and onloading of all the tasks related to the processing of the communication protocols. Therefore, it can be considered that p=1 in the LAWS model. As it has been shown, our HDL model allows the simulation of the CPU, the NIC, the network link, and all the effects of buses, bridges, caches, and main memory. Our experiments have been carried out with packets of 64 bytes generated by a synthetic packet generator and with different packet lengths in trace file provided by different benchmarks.

To simulate the protocol offloading that removes the protocol processing from the CPU, we have used the model shown in figure 2, when the CPU_0 is used for application processing and the NIC is used for protocol processing in order to take advantage of the parallelism between the communication and the application tasks.

To simulate onloading, CPU_1 is used to process the communication protocols instead of using either the NIC or CPU_0.

The behavior of the peak throughput improvement obtained in our experiments qualitatively coincides with the one predicted by the LAWS model when the parameter γ changes in the case of offloading and onloading. In our simulations, the value of σ verifies that $\sigma<\alpha\beta$, therefore; the maximum improvement happens when c=$\alpha\beta$ and it would be equal to $1/\alpha\beta$. Moreover, peak throughput improvement growth should be observed as $\alpha\beta$ decreases. Figure 6 and 7 show these effects for three different values of $\alpha\beta$.

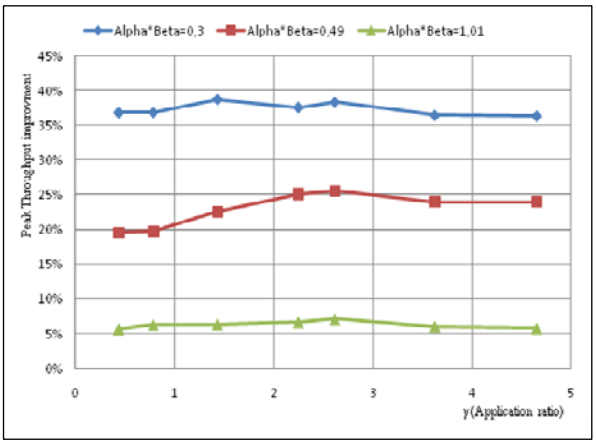

Fig.6 Peak throughput offloading with various values of $\alpha\beta$ and 1 Gbps.

Due to the effects of offloading in the network interface card, we can see that the higher network bandwidth the higher the peak throughput with $\alpha\beta$<1, as shown in Figure 7 compared with Figure 6. The LAWS model supposes that the communication path is pipelined throughout the processor, the NIC, and the network. Obviously, as these elements of the communication path share resources, the LAWS model only established an upper bound to the peak throughput improvement.

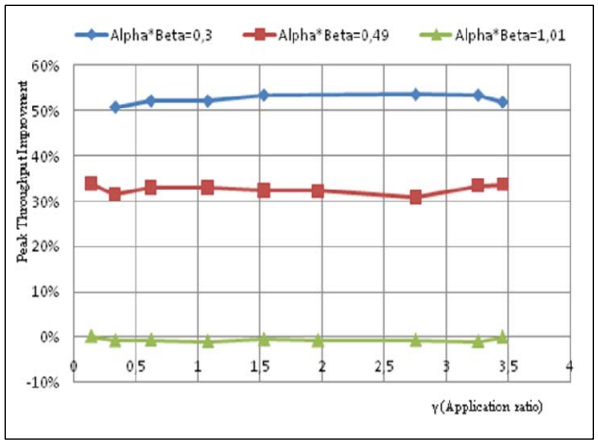

Fig.7 Peak throughput offloading with various values of $\alpha\beta$ and 10 Gbps.

Figure 8 shows the effect of onloading when $\alpha\beta$=1.15. In the case of onloading we cannot change the values of $\alpha\beta$ because: (1) the parameter α depends on the speed of processors (we have considered that all the processor cores have the same speed) and so α is always equal to 1; (2) the parameter β depends on the time delay on the NIC, but in this case, the driver is executed in the one of the processor cores and β will have a fixed value.

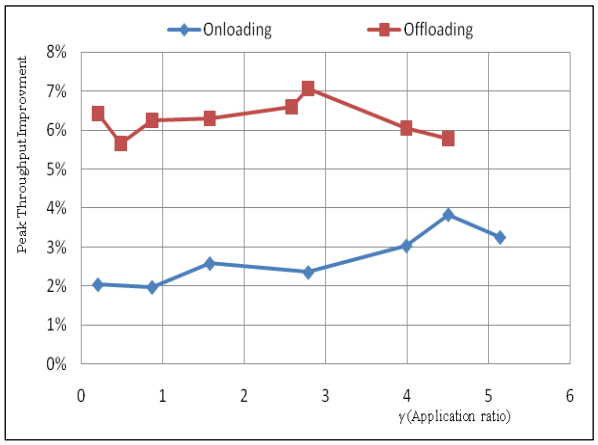

Fig. 8 Comparison of the peak throughput between offloading ($\alpha\beta$=1.01) and onloading ($\alpha\beta$=1.15), with 1 Gbps of link bandwidth.

Figure 9 compares the peak throughput between the base system, onloading, and offloading, while $\alpha\beta$=1.15 and 1.01 respectively. This test provides the network performance in terms of the available throughput. The same figure shows that the offloading and onloading alternatives provide improvements in the peak throughput, while these improvements depend on the value of the values of $\alpha\beta$.

Previous simulations have been done considering an average memory access time and no detailed cache simulation has been modeled. Figure 10, 11, 12 and 13 show the effect in the peak throughput improvement of including a two level cache in the simulation model.

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

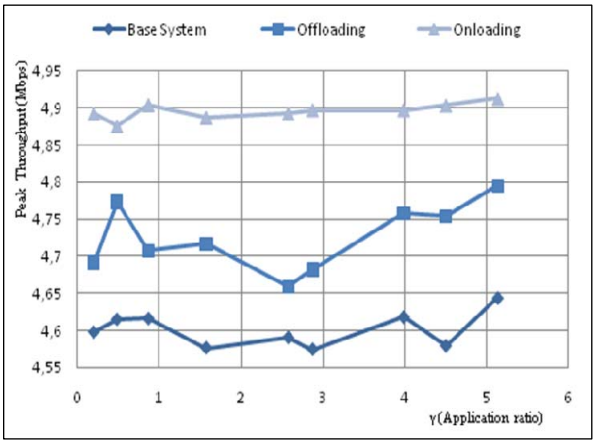

Fig.9 Comparison of the peak throughput between base system, offloading and onloading.

Also they show that caches improve the performance of the peak throughput provided by each network interface implementation alternative. Nevertheless the amount of the improvement is different between offloading and onloading. Thus, when the cache model is included, the improvements in base system and in the onloading model are bigger than the obtained with the offloaded network interface alternative. The behavior of offloading can be explained because, as the interface is mainly processed in the NIC, the use of a cache in the host does not affect in its performance improvement because the NIC uses its local memory for transferring data to the main memory. Figure 10 shows the improvement in the throughput in the case of offloading when L2 cache size increases with a specific size of L1, while Figure 11 shows the throughput improvement when L1 cache size increases with a specific value of L2.

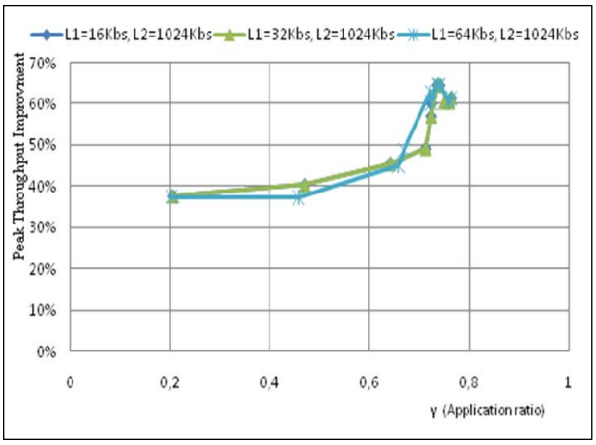

Fig.10 Comparison of the offloading peak throughput improvement with the effect of the L2 cache.

The onloading strategy provides better peak throughput improvement whenever the cache is available in the cores of the host processor as shown in Figure 12 and 13 that provide the peak throughput for different values of L1 and L2 respectively.

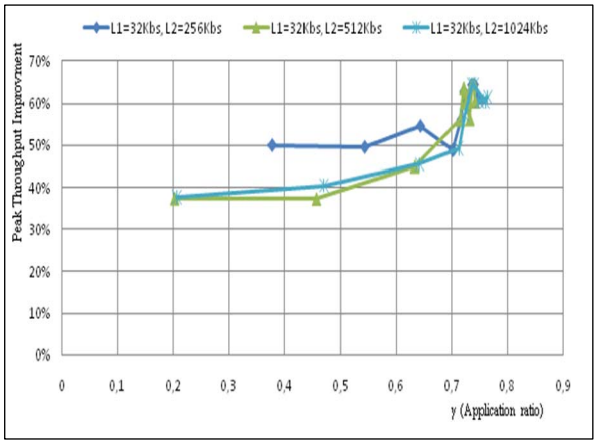

Fig.11 Comparison of the offloading peak throughput improvement with the effect of the L1 cache.

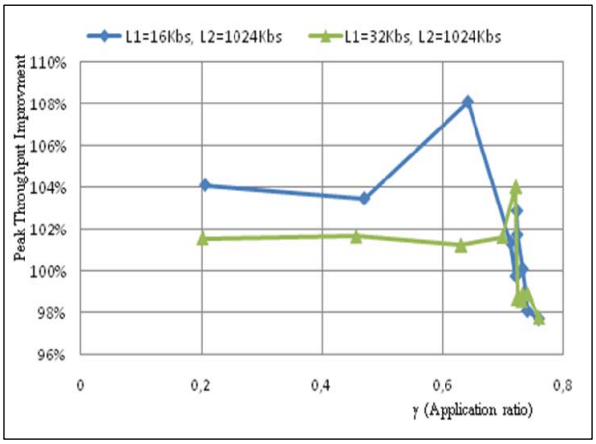

Fig.12 Comparison of the onloading peak throughput improvement with the effect of the different values of the L1 cache.

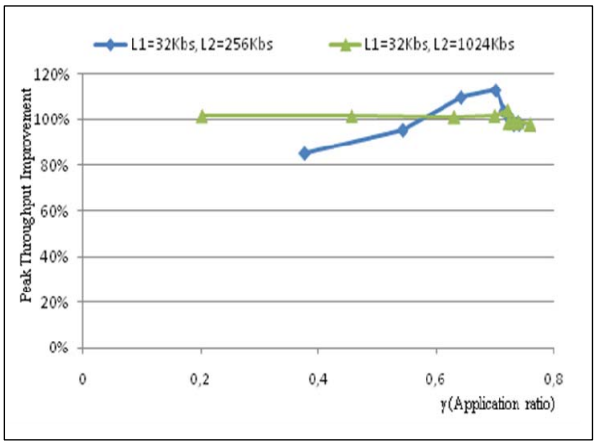

Fig.13 Comparison of the onloading peak throughput improvment with the effect of the different values of the L2 cache.

Moreover, the increases in the cache size produce throughput improvements especially in the case of onloading. The previous results have been obtained by using a generator that builds packets in the memory module of the sender. Nevertheless, we have also used traces corresponding to real

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

applications executed either in LAN or WAN environments. The file of traces includes Ethernet packets of different lengths (between 64 and 1518 bytes) and the time intervals to send them [19]. Figure 14 shows the peak throughput performance of offloading when we use this file of traces and different values of $\alpha\beta$.

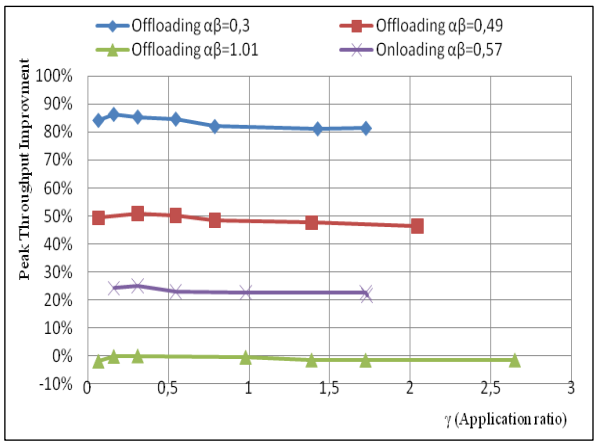

Fig.14. Comparison of the offloading peak throughput improvement with various values of $\alpha\beta$ and onloading (with a measured $\alpha\beta$=0.57).

The maximum value for the peak throughput improvement is reached when $\alpha\beta$=0.3. While, the worst case in our simulations occurs when $\alpha\beta$=1.01, because the NIC cannot take advantage of the offloading technique. In the case of onloading, when $\alpha\beta$=0.56, our onloading simulation provides peak throughput performance improvement reached to 25%. The offloading and onloading strategies do not affect only in the throughput reached but also the latency of the communication system. Figure 15 compares the behavior of the improvements in the latency corresponding to different values of $\alpha\beta$ in the case of offloading. It is also shown that for the maximum throughput improvement, when $\alpha\beta$=0.3, the minimum value of the latency is obtained.

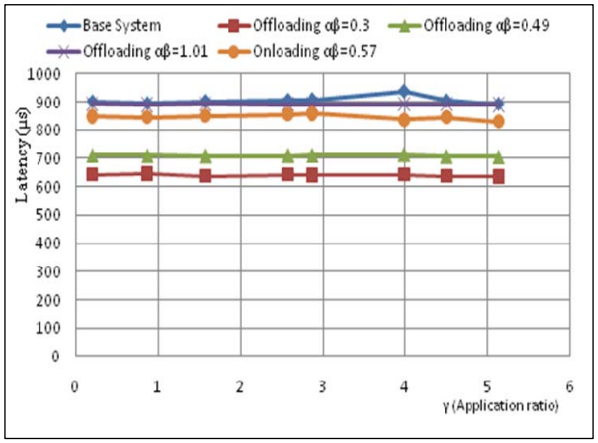

Fig. 15. Latency improvement with various values of αβ in the case of offloading and $\alpha\beta$ =0.57 in the case of onloading with 1 Gbps of link bandwidth.

The cache memory allows onloading to provide lower latencies than offloading. Figure 16 shows the effect of the cache in the latency improvement in the case of the offloading and onloading.

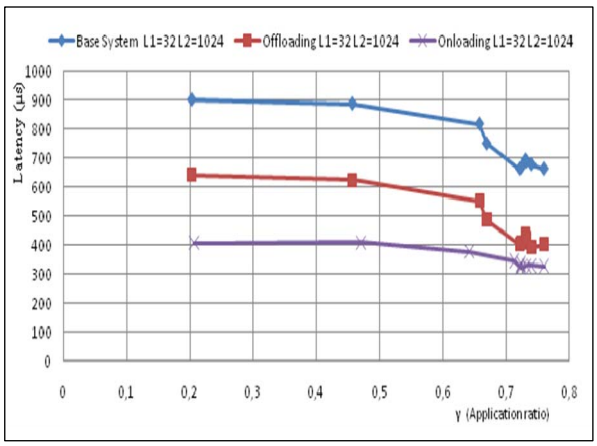

Fig.16 Cache effect in the latency with 1Gbps of link bandwidth.

Figure 17 shows the throughput improvement of the communication path with 1Gbps packet generator. The small improvement growth is observed as γ grows up to a specific value and after this value, the throughput improvement decreases.

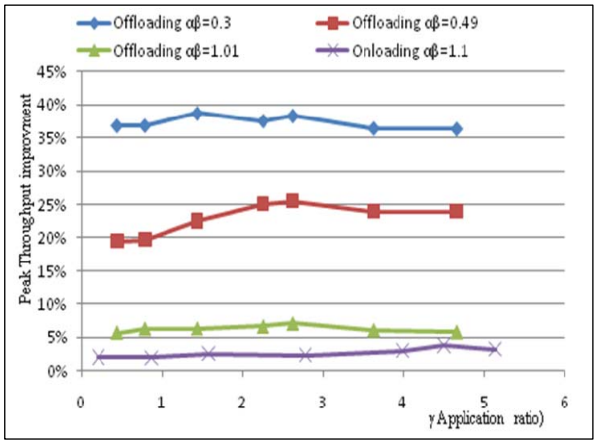

Fig.17 Throughput improvement of onloading and offloading with various values of $\alpha\beta$.

Figure 17 shows that there are important differences between the throughput improvement corresponding to each curve, not only in the amount of the throughput improvement but also in the location of the maximum and in the rate of the change in the throughput improvement with the application ratio. The reason for these differences came from the effects of the I/O buses, memory buses, etc.

ISBN: 978-988-17012-8-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

V. CONCLUSION

It is clear that the offloading and onloading strategies offer different advantages and drawbacks in the improvement of the communication path performance amongst different hardware/software configurations.

The HDL model which we have developed has made possible to describe and simulate the hardware and software components of the communication path and their interactions. Thus, our HDL model has allowed us the study of the offloading and onloding performance by controlling the parameters that affect in the behavior of the system. This way, an HDL simulation of the offloading and onloading alternatives, and the comparison of the experimental results obtained with the conclusions of the LAWS model, have allowed us to analyze the different elements that determine the performance of the network interface architecture and to check the efficiency of the approaches to optimize it. Our simulation results show that the onloading provides higher improvements than offloading in the conditions of our experiments. Moreover, the behavior which we have observed in the experiment results qualitatively coincides with the analysis and optimizations reached from the LAWS model. Therefore, it is possible to use the LAWS model to determine bounds for the experimental results.

The relative throughput improvement offered by offloading and onloading depends on the application rate workload to communication overhead of the implementation and on the packet size. Both strategies can be used to reduce the number of interrupt received by the host processor, while in the case of onloading the other processor is used to process all the communication tasks and NIC interrupts.

The simulation results show that changes in the LAWS parameters $\alpha\beta$ affect the improvement of the peak throughput in a similar way to that predicted by LAWS model. Nevertheless, the way in which the improvement peak throughput depends on the LAWS parameter γ in our experiments is different to what is predicted by the LAWS model. It is clear that the improvements predicted by LAWS are upper bounds of the improvements obtained by real systems, as it has been shown in our simulations.

ACKNOWLEDGMENT

This work has been funded by project TIC-1359 (Junta de Andalucia, Spain). The authors thank the referees for their useful comments.

REFERENCES

[1] D.L. Schuff, V.S. Pai, P. Willmann, S. Rixner, "Parallel Programmable Ethernet Controllers: Performance and Security", IEEE Network, 22-28, 2007,

[2] G. Ciaccio, "Messaging on Gigabit Ethernet: Some experiments with GAMMA and other systems", Workshop on Communication Architecture for Clusters, IPDPS, 2001.

[3] R.A.F. Bhoedjang, T. Rühl, H.E. Bal, "User-level Network Interface Protocols", 53-60, IEEE Computer, 1998.

[4] P. Balaji, P. Shivam, P. Wyckoff, D.K. Panda, "High performance user level sockets over Gigabit Ethernet", Proc. of the Cluster '02, 179-186, 2002.

[5] J. Chase, A. Gallatin, Y.G. Yocum, "End-System Optimizations for High-Speed TCP", IEEE Communications, 2000.

[6] H.W. Jin, P. Balaji, C. Yoo, J.Y Choi, D.K. Panda, "Exploiting NIC architectural support for enhancing IP based protocols on high performance networks", Journal of Parallel and Distributed Computing, 1348-1365, 65(11), 2005.

[7] K. Magoutis, M. Seltzer, E. Gabber, "The case against user-level networking", Third Annual Workshop on System Area Networks (SAN-3), 2004.

[8] D. Freimuth, et al., "Server Network Scalability and TCP offload, USENIX Annual Technical Conference", General Track, 209-222, 2005.

[9] H.-Y Kim, S. Rixner, "TCP offload through connection handoff", ACM Eurosys'06, 279-290, 2006.

[10] Y. Hoskote, el al., "A TCP Offload Accelerator for 10 Gb/s Ethernet in 90 nm CMOS", IEEE Journal of Solid-State Circuits, 1866-1875, 38(11), 2003.

[11] J.C. Mogul, "TCP offload is a dumb idea whose time has come", 9th Workshop on Hot Topics in Operating Systems (HotOS IX), 2003.

[12] D.D. Clark, et al., "An analysis of TCP processing overhead", IEEE Communications Magazine, 23-29, 1989.

[13] G. Regnier, et al., "TCP onloading for data center servers", IEEE Computer, 48-58, 2004.

[14] G. Regnier, et al., "ETA: experience with an Intel Xeon processor as a packet processing engine", IEEE Micro, 24-31, 2004.

[15] M. Rangarajan, et al., "TCP Servers: Offloading TCP processing in Internet Servers, Design, Implementation and Performance", Tech. Report, DCS-TR-481, Rugers University, 2002.

[16] B. Wun, P. Crowley, "Network I/O Acceleration in Heterogeneous Multicore Processors", In Proceedings of the 14th Annual Symposium on High Performance Interconnects (Hot Interconnects), 2006.

[17] P. Shivam, J.S. Chase, "On the elusive benefits of protocol offload", SIGCOMM'03 Workshop on Network-I/O convergence: Experience, Lessons, Implications (NICELI), 2003.

[18] M. Pirvu, L. Bhuyan, R. Mahapatra, "Hierarchical simulation of a multiprocessor architecture", Proc. Of the 2000 IEEE Int. Conference on Computer Design: VLSI in Computers and Processors, 2000.

[19] The Internet Traffic Archive, http://ita.ee.lbl.gov/html/contrib/BC.html.

[20] E.P. Markatos, "Speeding up TCP/IP: faster processors are not enough", in: IEEE21st International Performance, Computing, and Communications Conference, 2002.

[21] A.F. Díaz, J. Ortega, A. Cañas, F.J. Fernández, M. Anguita, A. Prieto, A., "Light weight protocol for gigabit Ethernet", in: Workshop on Communication Architecture for Clusters (CAC'03) (IPDPS'03), 2003.

[22] R. Westrelin et al., "Studying network protocol offload with emulation: approach and preliminary results", 2004

[23] C.L. Su, A.M. Despain, "Cache Designs for Energy Efficiency", Hawaii International Conference on System Sciences 28th, 1995.

[24] M5 simulator system Source Forge page: http://www.m5sim.org/wiki/index.php/Main_Page, 2009.

[25] M. Rosenblum et al., "Using the SimOS machine simulator to study complex computer systems", ACM Transactions on Modeling and Computer Simulation, 1997.

[26] P.S. Magnusson et al., "Simics: a full system simulation platform", IEEE Computer, 2002.