ATOLL – A Next Generation System Area Network

Jörg Kluge, Markus Fischer, Ulrich Brüning, Lars Rzymianowicz, Patrick Schulz, Mathias Waack
Department of Computer Engineering
University of Mannheim
B6 26, D-68159 Mannheim, Germany
email: joerg@atoll-net.de

Abstract
ATOLL (Atomic Low Latency) is the synonym for a new cluster technology centered around an ASIC chip design integrating all logic parts of a complete System Area Network. Besides its unique architecture ATOLL extends the hardware support for data transfer over the network to a limit, thus reducing communication latencies and enhancing concurrency.

This paper presents the chip’s key functionality – the mechanisms for the send- and receive-operations – in detail, but also illuminates some of the features newly invented with ATOLL. The latter cover such different items as an advanced interface for direct support of user-level communication, special locking modes allowing atomic access or techniques to clear network blocking – all intended to overcome the drawbacks of current SAN implementations.

Keywords: System Area Networks, Cluster Computing, Network Interface

1 Introduction

Currently cluster technologies are the only way to gain supercomputing power at affordable costs. This has led to an emerging market of so called System Area Network (SAN) solutions trying to address the special characteristics and needs of parallel computing. Compared to standard LAN implementations the improvements mainly concern a higher bandwidth, more sophisticated mechanisms to control data transfer and a better interface providing user-level communication or even a global address space.

In spite of these efforts, some serious problems still have not found a satisfying solution yet. One of the worst is the insufficient support of small message sizes, in which not only the bandwidth reaches less than a fraction of the proclaimed peak value but also the latency of each single transfer becomes significant. This is a grave situation, as parallel applications often rely on communication patterns with rather small data packets. For example, a recent study [8] examining a representative cross-section of various programs resulted in heavy quotas for sizes between 16 bytes and one kilobyte.

It is quite obvious, that small messages suffer disproportionately from all operations causing a static overhead during transaction and that their poor performance originates to the greater part from here. Hence, there have been several attempts to reduce the amount of software layers passed during a transaction resulting in lower latency and better bandwidth on bypassing the operating system completely. Avoiding the time consuming kernel traps implies giving the user process full control of a network device – despite probable security doubts this also expresses the absence of any higher instance managing the multiplexing of such a device. Nevertheless, as recent research projects (U-Net, BIP, HPVM) have shown, the advantages of this concept often turn user-level communication into an essential key mechanism [2].

However, all achievements made in streamlining the software interface only alleviate but do not overcome the unsatisfying bandwidth characteristics. One reason is the costly multichip architecture of today’s SAN interfaces. A closer look at common solutions, like Myrinet or Dolphin’s SCI, discloses a complex structure with on board microcontrollers, autonomous DMA engines, SRAMs, additional programmable and external switching logic - each interacting in low concurrency with the other components and thus producing noticeable overhead at each operation. In processor technology comparable design approaches with almost similar problems have been outdated with the upcome of the RISC philosophy. In analogy to the achievements made here, the key demands for a next generation of SANs, therefore, should include less components through high integration scale, a fully pipelined design, highest possible concurrency and the hardware implementation of common and frequently used functions.

Another critical item arises from the dedicated hardware structure found in cluster topologies. In contrast to tradi-
tional supercomputers all computing components work autonomously fully equipped with their own operating system, applications, hardware and even power supply. At first sight the redundancy and loose coupling of most parts seem to increase reliability. But clusters do not take advantage of this as they usually lack any instance supervising and managing system sanity as a whole. The main cause for this is the network not only being non-redundant and failure sensitive but in terms of managability also an encapsulated add-on part. So the interconnecting element eluding from effective control creates a situation, where the top-sight shows a conglomeration of individual host machines instead of an integrated system. Hence, having most diagnostics restricted to the single host (with the opportunity of posting the results to some kind of cluster management server) clusters cannot benefit from their basically fault tolerant architecture and even tend to be more sensitive to malfunction because of the lower quality standards in the commodity-off-the-shelf market. Providing each – basically ‘intelligent’ – host system with more opportunities to control the network activities would weaken the problems. However, this requires a tighter coupling between computer and network hardware where all network components must be directly accessible by the host.

The ATOLL architecture for a new SAN was consequently developed to fulfil the demands listed. This means ATOLL truly initiates a radical design approach extending the hardware integration to a limit with all network logic closely attached to a host computer.

Introducing the hardware design in the next section, an overview of ATOLL’s chip architecture and cluster topology will be given. After this the implementation of one of ATOLL’s most important and enhanced functions in hardware, the message handler for autonomous data transfer through the device, will be discussed in detail in section 3. Thus provided with a rudimentary understanding of the transfer mechanisms a description of the on-chip analysis and control structure follows in section 4, which enables an adapted host not only to enquire the network status but also to intervene in case of problems. At last, section 5 presents some performance data taken from a simulation model.

2 Architectural Overview

An ATOLL network encloses only the network interface cards (NICs) and the interconnecting cables. The cards support both, standard 32 bit/33 MHz and enhanced 64 bit/66 MHz (or any combination of this), PCI modes. One single chip, which contains the whole network logic, is mounted on each card. Central and most striking component of this ATOLL chip is a 4 × 4 bidirectional (i.e. 8 × 8 unidirectional) crossbar switch connecting 8 surrounding ports (see fig.1). Serving host and network side equally, half of them, the host ports, are used to provide four independent physical network interfaces for a host system whereas the others, the link ports, define the links to four neighbouring nodes.

Hence, adapted to a host system, a single ATOLL NIC shows up as four devices. Of course, the network traffic of all of these devices has to be serialized on the PCI bus. Capable of 528 MByte/s in the latest available revision [9] already one of the network interfaces with its 2 × 250 MByte/s (bidirectional) bandwidth can flood the bus. So concerning the data throughput, advantages only arise from a better average bus load in contrast to the short-term utilization of resources by a single device. For best results, the chip’s integrated PCI interface also combines transactions to bursts whenever possible.

Far more important and thus being the main reason for the sophisticated implementation is the enhanced support of user-level communication on 2/4-way SMP nodes. Each active process on one of the multiple CPUs can gain direct access to its own communication device. As SMPs tend to become the computing backbone of more and more clusters this improvement overcomes solutions, where hosts had to be equipped with multiple NICs in the past [7].

Nevertheless, in case an ATOLL device has to be shared care has been taken to keep its context very small. Additionally, the belonging data structures reside completely in main memory and are only referenced by pointers through the chip. Thus, apart from the necessary operating system kernel invocation context switches are very cheap to handle.

![Figure 1. ATOLL Chip Block Diagram](image-url)

The network ports between host ports and crossbar logically separate the chip’s host and network partition. The host’s internal 64 bit data stream is converted here to the byte-parallel 9 bit network format and vice versa. Messages from the adapted host ports are separated into 3 frames (route, header, data) and provided with special control characters. The frames themselves are broken down to 64 byte units, each protected by a CRC.

The link ports adjacent to the crossbar drive outgoing and receive incoming 9 bit LVDS signals. The chip runs at 250 MHz clock frequency, parallel data transmission is re-
alised at the same rate over a distance of up to 10 m. A single ATOLL NIC therefore provides a total bisectional bandwidth of $4 \times 2 \times 250 \text{ MByte/s} = 2 \text{ GByte/s}$. As potential CRC errors are most likely to occur on the cables between the hosts, each individual 64 byte unit is preserved in a buffer register of the link port until it is accepted by the preceding stage. In case of an error a retransmission is invoked [1]. As ATOLL does not trash any data, this mechanism guarantees the delivery of each message passed to the network.

With the crossbar connecting arbitrary ports, messages can be routed from the host to the network and vice versa. However, providing the full functionality of a switch in the conventional sense, also 2 link ports can be coupled – routing the message through the network (fig.1). To achieve low fall-through times source-path routing is combined with cut-through switching.

The ATOLL architecture basically fits very well to all topologies connecting a host to four neighbours, like 2d-Meshes (see fig.6), Hypercubes, etc. However, this is no direct restriction, as it is easy to increase the total amount of links per host system with additional NICs.

3 Message Handlers

One of ATOLL’s most advanced features are continuously active receivers for incoming messages – capable of spooling data autonomously and in full concurrency to the processor into the host’s memory. ATOLL implements the functionality of such a message handler completely in hardware as DMA engine. For scalability and performance, one of them exists for each host port.

Additionally a software interface for data transfer via PIO is provided with ATOLL’s API. It also acts as a kind of message handler – with the difference that no concurrency can take place.

To outline the benefits of both, the steps of a send/receive scenario in DMA and PIO mode will be discussed in the following.

3.1 Hardware Message Handler

Sending a DMA message involves the interface to require information on the receiver and the memory location of header and data, both coupled with a length for the appropriate block structure. The look up in the routing table is organized in the same manner [6], which means everything needed can be stored in a small descriptor (fig.2) containing pointers and length. All data, information and tables reside in user-space. They are therefore directly accessible without any time consuming calls like kernel traps.

In this context, first the ATOLL API will transfer data into the provided DMA memory region which has been pinned down when initializing the ATOLL device. This segment can be seen as a circular ring buffer that is controlled via read and write pointers. Secondly the descriptor is built.

$\begin{array}{|c|c|}
\hline
\text{Routing Offset} & \text{Routing Length} \\
\hline
\text{Header Offset} & \text{Header Length} \\
\hline
\text{Data Offset} & \text{Data Length} \\
\hline
\text{Tag} & \\
\hline
\end{array}$

4 x 64 bit or 1 Cacheline of 32 Byte

Figure 2. DMA Message Descriptor

By increasing the descriptor table write pointer, the hardware is triggered and consumes the descriptor. This way, the hardware message handler takes over and actually delivers the message. ATOLL therefore provides an atomic message startup by this single write.

When the descriptor has been read, the hardware increments the descriptor read pointer, which frees this entry in the descriptor table. Then the message will be taken out of the provided location. To implement broadcast functionality to hardware, the user can decide, whether the data region read pointer will be increased, which frees this space, or not, which makes broadcasts possible without writing a message twice to the main memory. ATOLL will read descriptors until no further messages are available, which is indicated by equal descriptor table write/read-pointers. These steps are depicted in more detail in figure 3.

Figure 3. Steps Sending a Message

On the receiving side the corresponding descriptor is directly assembled in hardware and will be stored in main memory user space to be accessible by the software (fig.4).

3.2 Software Message Handler

When sending in PIO mode, only a small overhead appears and allows the best performance for short messages; however, concurrency cannot take place. Recent research
aims at providing zero mechanisms in which the message is
directly taken out of application memory and injected into
the network. On the receiving side this data is again directly
stored in a user’s application memory. This has the benefit
of no intermediate buffering.

ATOLL’s PIO mode actually offers this functionality. An
important feature coming with PIO and therefore involving
the CPU is that MMU functionality is provided. The reason
is quite obvious. With hardware residing on the other side
of the PCI bus, memory entries have to be cached to be able
to spool data to the correct destination. Usually this location
will change during runtime. But also memory segments will
be paged by the OS and therefore entries need to be updated
unless memory is pinned down. However, this is a time-con-
suming procedure involving the interaction of the system.
For small messages or barriers PIO is also advanced since
a direct store from the CPU cache to PCI memory location
can take place. With the utilization of chipset features like
‘write combining’ and ‘read prefetching’ [4] performance
has been further improved.

In case a matching receive is not posted, the FIFO buffers
on the appropriate host port run full with the message tail
eventually blocking further instances of the network. There-
fore, an interrupt will be invoked by the hardware, and soft-
ware will be activated by calling a programmable signal
handler function. This would empty the buffers and store
the data in memory to be fetched later.

On message receive, the hostport decides upon the mes-
sage length and an adjustable threshold register whether to
use PIO or DMA mode [10].

3.3 Methodical Benefits

Message exchanges using message handlers have already
demonstrated to improve efficiency, to lower latency
and to increase bandwidth respectively. Software message
handlers were also exploited in recent research activities
such as AM[11] and U-Net[12]. Therefore, ATOLL relies
on well known and proven mechanisms for data transfer. As

Figure 4. Steps Receiving a Message

4 Enhanced Analysis and Control Facilities

Special Registers are also provided to gain transparency
of the actual operations on the network with the option to in-
tervene current message transfer. Here ATOLL profits from
its high grade of integration with all parts of the network
logically connected to a host system, thus encouraging
and belonging routing information [6]. Due to this design
decision a misfit between topology and routing possibly
leads to the occurrence of deadlocks, which therefore cannot
be excluded entirely. The usual way to handle the problem
is the implementation of virtual channels [3] enhancing the
amount of logical paths with additional buffers in front of
each crossbar switch. As their number is still limited due
to the amount of installed buffers this is not a general so-
lution covering all cases. Furthermore, the buffers share a
physical path and, thus, need to be multiplexed – enlarging
complexity and contradicting ATOLL’s design philosophy.

The other solution for handling deadlocks by quashing
single messages hardly needs to be mentioned because it
results in an irrecoverable loss as no copies are held.

Hence, without restricting the possible topology config-
urations unnecessarily, the desirable abilities for network
control become an essential feature to solve message block-
ning.

4 Enhanced Analysis and Control Facilities

A major improvement is that host memory also is a far
less scarce resource compared to on-board SRAM. There-
fore a lot more buffer space is available. In contrast NICs
with on-board storage memory are susceptible to overflows
– especially if an incoming message is not directly fetched
by a matching receive. The consequences have already been
described for the comparable situation of ATOLL running
out of buffer space in PIO mode – with the difference of
currently known NICs ignoring the problem.

Another decision was to add user level locking mechani-
isms – protecting critical operations like an increment on
the write descriptor – to support multithreaded applications
(e.g. one user’s threads sharing a single device). This was
necessary, because processors such as Intel’s Pentium III [5]
require a trap into the system to access semaphores. For an
efficient user-level implementation the functionality had to
be recreated, which means ATOLL provides registers with
test+set functionality for each hostport.

4 Enhanced Analysis and Control Facilities

4 Enhanced Analysis and Control Facilities

Special Registers are also provided to gain transparency
of the actual operations on the network with the option to in-
tervene current message transfer. Here ATOLL profits from
its high grade of integration with all parts of the network
logically connected to a host system, thus encouraging
and belonging routing information [6]. Due to this design
decision a misfit between topology and routing possibly
leads to the occurrence of deadlocks, which therefore cannot
be excluded entirely. The usual way to handle the problem
is the implementation of virtual channels [3] enhancing the
amount of logical paths with additional buffers in front of
each crossbar switch. As their number is still limited due
to the amount of installed buffers this is not a general so-
lution covering all cases. Furthermore, the buffers share a
physical path and, thus, need to be multiplexed – enlarging
complexity and contradicting ATOLL’s design philosophy.

The other solution for handling deadlocks by quashing
single messages hardly needs to be mentioned because it
results in an irrecoverable loss as no copies are held.

Hence, without restricting the possible topology config-
urations unnecessarily, the desirable abilities for network
control become an essential feature to solve message block-
ning.
4.1 Status Register Set

The central element of an ATOLL-chip is the crossbar, connecting pairs of individual ports. So naturally this is the ideal instance for gathering information about state and progress of all messages running through the chip.

For this purpose small buffers have been implemented, keeping track of all requested and granted connections completed by the actual data passing. Putting it all together, this results in one 32 bit register (aligned to 64 bit in PCI address space) for each of the 8 ports, consisting of 8 bits (1 out of 8 code) for requested plus 8 bits for granted channels, 9 bit data and some additional bits concerning data flow (see fig.5). Data is tapped at the appropriate port before entering the crossbar. Reading the status registers has to be done by

addressing them directly in PIO mode. Therefore, bus latencies would normally inhibit a continuous selective analysis, which means transfer has to be slowed down by external control. To circumvent the whole crossbar coming to a stall, the special modes providing the requested functionality only affect dedicated connections using the port in front of the crossbar:

- A **snapshot** returns the contents of a status register.
- Each port can be set to a **freeze mode**, holding all operations and data transfer.
- Each time **single step** is invoked a frozen channel is released for one clock cycle.
- A **consuming read** is used to move data stepwise to the host system. The data is not propagated to the subsequent crossbar.

All operations are invoked by writing to a control register, which is shown in fig.5.

4.2 Solving Network Blockings

As ATOLL uses a combination of cut-through and wormhole routing, messages have a prefix containing a string of routing words. Each of the words defines the port number of a node. A message being routed through the network always requests the next outgoing channel (e.g. port) from every crossbar on its path by its leading word, which is cut-off thereafter. As a consequence the history of the taken path is lost [10].

The *consuming read* functionality provides a basic mechanism to tear off messages from the network and thus principally enables the adapted host system to clear a hazardous situation caused by stuck data on the ATOLL NIC. However, regarding the above mentioned routing strategy, this can only be done at the head of the ‘worm’ without corrupting the whole message. Solely under these circumstances a torn-off and buffered message kept in the host’s main memory still holds the complete routing information needed to find the remaining path to the target node and thus can be reinjected to the network.

Fortunately the occurrence of stuck data that cannot be seized by its routing head is limited to severe node or link failures (e.g. power down, disrupted cable, etc.), where the remaining pieces of a message cannot be recovered and have to be trashed anyhow. All other blocking situations result from one message requesting the crossbar arbiter for a port already granted to another. This can easily be manifested by a **snapshot** on the status registers, showing at least two ports with the same request-bits with only one of them having the appropriate grant.
Performance estimations have been extracted from the simulation of the RTL implementation of the ATOLL chip together with a PCI bus functional model. The two-node testbed extracts the time between the PCI-bridge of a sender and a receiver host, both directly connected via ATOLL. The one-way hardware latencies hereof start at a minimum value of 1.4 µs (for a 16 byte message) [6].

Measurements of the link utilization shown in fig.7 – by means of the pure data payload running over the interconnect wire between two ATOLLs – round out good latency values. Outstanding performance is attained for small messages, starting at 100 MByte/s for 32 Byte data packets and already saturating at a size of 128 Bytes with approx. 225 MByte/s or 90% physical bandwidth. Of course these values do not describe the performance of a real system at software level, as they do not consider significant latencies from the software overhead or the bus system. In spite of this drawback, the data nevertheless states the high efficiency of the raw ATOLL hardware.

Additional simulations show a delay of 18 clock cycles for sending a message across a node (e.g. from one network port to another), which means 72 ns per hop. Compared to the data above, even the sum over a larger amount of hops will make a negligible contribution to the total latency. Therefore, connecting nodes over longer paths principally remains uncritical, as long as blocking does not occur.

Conclusion

In this paper we have introduced the ATOLL architecture with its radical new design for a System Area Network. High integration scale and the sophisticated implementation of the main data transfer mechanisms with message handlers in hard- and software endorse noticeable speedup. This could be stated by the presented performance curve taken from a simulation model, where particularly the critical data sizes show outstanding bandwidth. Furthermore, the tight physical connection of the network card to a host system facilitates new techniques for controlling system sanity.

Putting it all together, we believe ATOLL makes a step ahead of today’s technology standards – therefore being a true candidate for a next-generation SAN.

References