Regular Research Paper (RRP) for

1999 International Conference on
Parallel and Distributed Processing Techniques and Applications
(PDPTA'99)

Technical Session:
Cluster Computing Technologies, Environments, and Applications
(CC-TEA)

Paper ID: 780SP

ATOLL: A Network on a Chip

Lars Rzymianowicz, Ulrich Brüning, Jörg Kluge, Patrick Schulz, Mathias Waack
Chair of Computer Architecture, University of Mannheim
B6, 26, 68159 Mannheim, Germany
email: larsrzy@atoll-net.de

Abstract

Current System Area Networks still have a few drawbacks. Firstly, the price/performance ratio is too high for so-called low-price Cluster Computing. Secondly, the trend to use SMP machines with two or more CPUs as cluster nodes puts a lot of pressure on a single Network Interface (NI). Thirdly, almost all current implementations need a relatively large message size to deliver about 50% of their physical bandwidth.

ATOLL addresses these problems through several unique characteristics: it combines four independent NIs, an 8x8 crossbar switch and four links in one chip, thus lowering costs significantly. Also, data can be transferred via Programmed I/O or DMA, thus giving the programmer the greatest flexibility. And the NI context is fully loadable, so the concept of virtual NIs (like specified in the VI Architecture) can be applied.

Keywords: System Area Networks, Cluster Computing, Network Interface

1 Motivation

Cluster Computing based on PC Commodity-off-the-Shelf (COTS) technology has become one of the hottest topics in the field of High Performance Computing (HPC) over the last five years. To compete with traditional supercomputing technology, two important components are needed: powerful COTS workstation nodes and high performance System Area Networks (SAN). While efficient and pricable cluster nodes are already available due to today's Intel P6-based PCs and Alpha-based workstations, SAN technology is still evolving and relatively expensive.

In the last years, several SANs have pushed PC-based Cluster Computing ahead, namely ServerNet, Myrinet, SCI, Memory Channel [1], and Synfinity [2].

A closer look at various performance measurements shows that several SAN implementations need a relatively large message size to deliver an appropriate bandwidth. The typical bandwidth curves start with moderate values and begin to rise at message sizes of 4-8 Kbyte. But a large part of the communication traffic in message passing applications consists of rather small messages (8-64 bytes). So the efficient handling of small messages is a critical issue for SANs.

The above observation can be tracked down to architectural characteristics, which prevent the SANs from achieving a better performance for small chunks of data. The intermediate copying of data into an extra SRAM on the NI card, slow NI microcontrollers, and the lack of a processor interface to the NI for Programmed I/Ocontrolled data transfer can prevent the efficient transfer of small messages.

Another critical issue is the lacking support for SMP machines as cluster nodes. Processes running on the same SMP node can gain from the greater communication bandwidth via shared memory, if supported by the protocol layer. The use of SMP machines will continue to grow with their broad availability. But a single SAN adapter quickly becomes a bottleneck, when used inside a four- or eight-CPU cluster node [3].

Currently, overall costs are still one of the main reasons which prevent the broad use of SAN technology. The current price gap between low-cost LANs and high-performance SANs is too large. For most SANs, the ratio between node and interconnect costs is almost 50:50. Of course, low-volume production is one factor, but also architectural characteristics (multi-chip solutions, expensive on-board SRAM) result in high prices.

The ATOLL [1] SAN tries to address all these problems. Costs are lowered through a tight integration of all components (including the switch!) into one single chip. To provide better support for SMP machines, the NI is replicated four times and its context can be switched by software. The overall architecture of ATOLL is described in section 2. The data transfer mechanisms are discussed in detail in section 3. Finally, section 4 presents first performance measurements.

2 The ATOLL architecture

One primary goal of ATOLL is to close the price/performance gap between low-cost LAN technology like Fast Ethernet and current SANs, which cause costs around $1500 for an NI and $2000+ for a switch. These high costs are due to the multi-chip solutions and additional on-board SRAM.

To lower costs significantly, ATOLL integrates the NI and the switch into one chip. An 8x8 unidirectional crossbar switch and 4 links eliminate the need for any additional switching hardware. To efficiently support SMP nodes, four independent NIs are integrated. A 64 bit/66 MHz PCI bus interface completes the design. This compact *Network on a Chip* solution enables low cost parallel computing at a yet unmatched price/performance level. To boost the communication performance and keep up with the fast design cycles of today's microprocessors, the ATOLL chip is implemented in a state-of-the-art 0.25 μm technology with a clock frequency of 250 MHz.

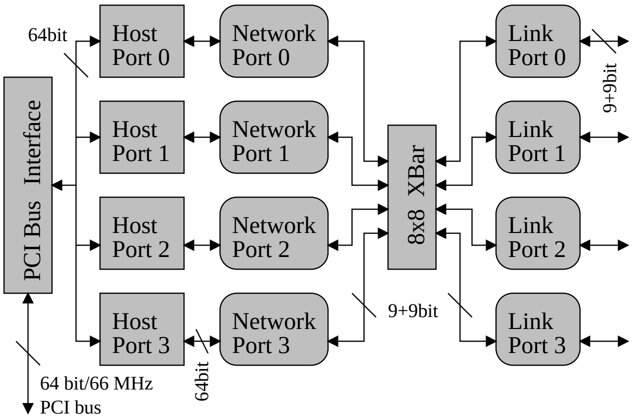

Figure 1: ATOLL chip block diagram.

The top-level block diagram of the ATOLL chip is shown in Figure 1. In the following, each basic block is listed with its main tasks, whereas the focus is on the Host Port, which serves as a processor interface to the ATOLL network.

[1] www.atoll-net.de

2.1 PCI bus interface

The PCI bus interface implements a fully compliant 64 bit PCI v2.1 interface, capable of running at up to 66 MHz. This decision was made regarding several aspects:

1. a 32 bit/33 MHz PCI implementation would become a major bottleneck, since today's system busses and the targeted bidirectional link bandwidth of 500 Mbyte/s offer much higher data rates

2. 64 bit/66 MHz PCI slots are already available in workstations and will be a standard component of COTS PCs in the near future

The PCI bus is the only system-independent I/O interface accepted by the computer industry and thus allows ATOLL to connect to a large range of systems.

2.2 Host Port

The Host Port can be viewed as the interface to the ATOLL network. It offers a processor interface to send and receive data via Programmed I/O and DMA to or from the network, which is described in detail in section 3.

2.3 Crossbar, Network and Link Port

The Network Port is responsible for converting the 64 bit data stream from the Host Port into the byte-parallel ATOLL network format and vice versa. A message is divided into three frames: routing, header and data. The Network Port inserts special control bytes to ensure the correct framing. Each message is broken down into multiple 64 byte link packets, which are protected by a trailing CRC.

Figure 2 depicts the general format of one message. To prevent buffer overflow, special control bytes for reverse flow control (STOP, CONT) are inserted into the data stream on the link level. A special feature is the breakdown into 64 byte link packets. These packets are buffered inside each Link Port and can be retransmitted in case of a CRC failure on the receiving side. The link packets are released through the exchange of acknowledge control bytes. A detailed description of the retransmission mechanism and the overall link protocol can be found in an earlier article [4] about a predecessor of the current ATOLL architecture.

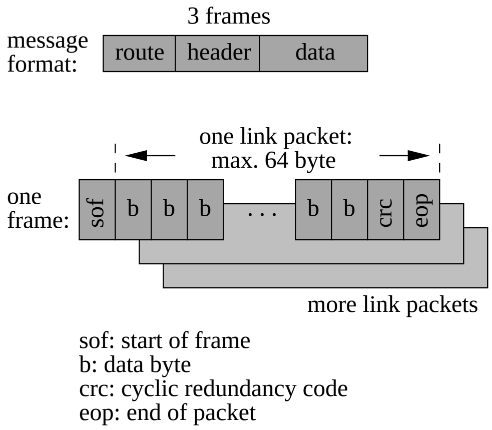

Figure 2: ATOLL message format.

Finally, the on-chip crossbar connects four independent Network Ports to four Link Ports via full-duplex links. ATOLL uses source-path routing to keep the latency as low as possible. This enables the design of a crossbar architecture with a fall-through time of 6 clock cycles, resulting in impressive 24 ns.

3 Data transfer

The implementation of the data transfer mechanisms offered by a NI is the most decisive point in SAN design. Several conditions have to be considered:

- a lot of parallel applications tend to communicate only small data packets at a high frequency, like e.g. fine-grain applications with a global synchronization after each iteration step
- single cycle accesses to the same address in the I/O space result in a very poor performance and should be avoided. Today's memory systems, from the processor's bus interface through the chipset and the memory controller down to the PCI bus, are optimized for linear transactions. Modern microprocessors include write buffers to assemble several stores into one burst transaction, chipsets [5] offer also write-combining and read-prefetching buffers, and the PCI bus reaches nearly its peak bandwidth when transferring data in long burst cycles as busmaster

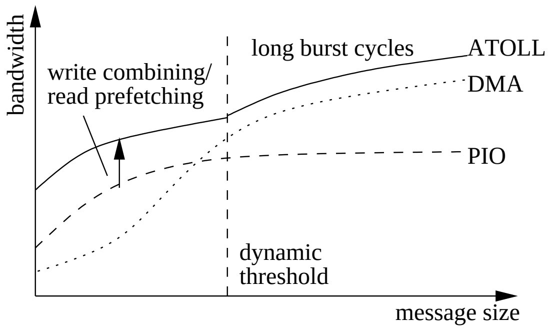

Figure 3: Typical bandwidth of PIO and DMA.

There are two possibilities: Programmed I/O means the processor copies data word for word into the NI. Or DMA, where the processor only sets up a job descriptor pointing to the data, which is then loaded and processed by the NI itself.

Figure 3 depicts the normal bandwidth curves of both modes. Recent studies [6] have shown that the use of techniques like write-combining for PIO-based transfer can significantly boost the data transfer performance from the main memory to the NI, even outperform DMA for each message size. ATOLL assembles the best features of both methods. The NI offers PIO- and DMA-based data transfer.

3.1 High-speed Programmed I/O

The PIO interface consists of one 64-stage FIFO in either direction (send/receive) and some control and address mapping logic. To enable write-combining for sending message data, the CPU recognizes the Fifo as an address region, which is separated into different areas according to the three message frames, as depicted in Figure 4.

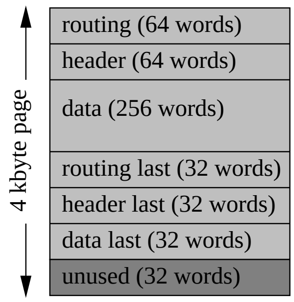

Figure 4: Address layout of the PIO-Send FIFO.

The size of each address region is specified in brackets (64 bit words). The address areas are aligned on cacheline boundaries, and small separate areas are used to mark the last dataword of each frame. With write-combining enabled, linear stores to this area will result in assembling data inside the write buffer, from where it will then be propagated in one burst cycle to the NI, as shown in Figure 5.

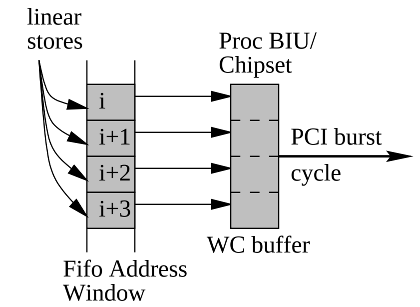

Figure 5: Enabling write combining.

In a similar manner, a second FIFO for receiving messages is visible to the processor. It has been made sure that single cycle transactions with a poor performance will be avoided if possible.

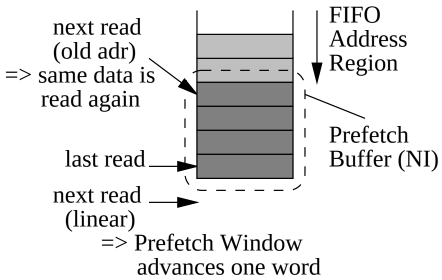

Figure 6: A prefetchable FIFO.

Read-prefetching is a technique of modern memory controllers and I/O chipsets to enhance the bandwidth for consecutive loads. The problem is that all these mechanisms are designed for side effect-free memory transactions. A prefetch buffer located in the PCI chipset of the host may drop data due to a context switch. In this case the new process overrides prefetched data, which have not yet been consumed by the receiving process. If the data are simply shifted out of the FIFO during the first access, wrong data will be delivered with the following load of the same address.

To deal with this situation, ATOLL implements a small prefetch buffer, which keeps the most recently read data of the FIFO. Since the processor issues a linear sequence of load addresses to read message data from the NI, the above situation can be detected through a simple address comparison. If a read address points to the same location as the previous one, prefetched data will be read again. Both possibilities are shown in Figure 6.

3.2 DMA-based communication

Two data regions for DMA-based message transfer are located in the main memory. One large memory block is used as a buffer for the routing table, message header and data. The other area is used as a descriptor table, which is composed of fixed-size (1 cacheline) entries.

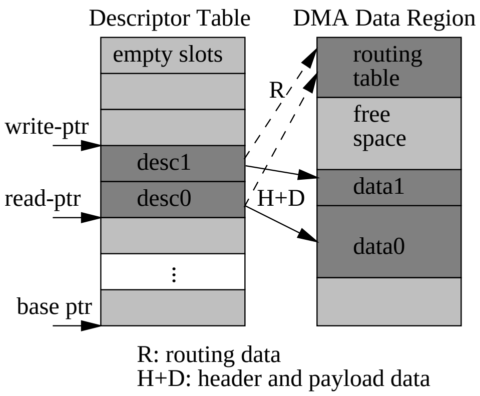

Figure 7: Data structures for the DMA mode.

A descriptor describes a DMA job for the NI and consists of three pointers to the data region (one for each message frame), together with their length. The fourth 64 bit dataword is used as an additional tag by the protocol layer. Figure 7 depicts both memory blocks on the sending side with two jobs enqueued. The same data regions, except the routing table, reside on the receiving side.

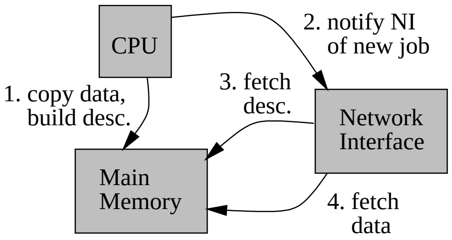

Figure 8: DMA-based message send.

Figure 8 visualizes the basic steps of a DMA-based send operation:

1. the CPU copies the message data into the pinned DMA memory area. Since the NI deals with physical addresses (lack of MMU) this intermediate step is necessary. A job descriptor is added to the descriptor table

2. the NI is notified of a new job entry via updating the write pointer of the descriptor table inside the NI

3. the NI fetches the job descriptor from the main memory

4. the NI successively fetches routing, header and payload data of the message and forwards them to the ATOLL network

Besides this basic mechanism for the DMA-based data transfer, ATOLL implements a number of unique features to enhance message passing operations. To support multicast send operations, the data regions can be referenced by multiple job descriptors. This reduces the copy overhead of the processor down to a single copy, instead of one for each destination. The CPU only needs to build several job descriptors, which send the same data to many different recipients. The last descriptor deallocates the data region.

The context of the NI (base address for memory areas, pointers, etc.) is accessible from the CPU. The NI can be multiplexed between several users by simply disabling the NI, switching to another context and enabling the NI again. This mechanism is a key factor to the concept of virtual NIs, as specified in the VI Architecture [7].

3.3 Control transfer

Another critical design decision is the notification of the processor in case of a completed message transfer. Current implementations rely on an interrupt-driven notification or on polling. One solution would be that the NI issues an interrupt after a message has been written completely to the main memory. But interrupts are an expensive OS service and increase the latency of a receive operation significantly. Another solution would be to poll a status register of the NI. But polling waists bus bandwidth and can degrade the performance through the interruption of burst cycles on the I/O bus.

To avoid such drawbacks, the ATOLL NI mirrors the significant status registers into the main memory (FIFO status, message completion, current transfer mode, etc.). The update frequency of this status area is configurable. This mechanism enables the processor to poll on cache-coherent memory, thus avoiding any read cycles on the I/O bus. We expect a significant performance gain by this feature.

4 Performance estimations

First performance estimations have been extracted from the simulation of the RTL implementation of the ATOLL chip. The two-node testbed consists of a PCI-Master model (acting as CPU/chipset), two ATOLL chips connected via all four links and a PCI-Slave model on the receiving side (acting as main memory). Table 1 depicts the one-way latency of messages, from the store of the first word until the message is completely received via DMA in the second node. These numbers do not include the arbitration of the system bus and the data transfer from the PCI bus to the main memory and vice versa on both sides. This will add additional 1-2 μs to the latency.

Table 1: One-way latency of small messages

payload (byte)	16	32	64	128	256
latency (μs)	1.4	1.6	2.0	2.5	3.9

More detailed performance values will be presented on the conference.

5 Acknowledgements

We would like to thank the PCS Group of Siemens, Augsburg for their excellent collaboration, especially in the physical design flow. We would also like to thank Markus Fischer for his comments on the paper and the ATOLL API. Last but not least, Berthold Lehmann implemented the Host Port of the ATOLL chip.

References

[1] M. Fillo and R. B. Gillett. Architecture and implementation of MEMORY CHANNEL 2. *Digital Technical Journal of Digital Equipment Corporation*, 9(1):27–41, August 1997.

[2] J. Larson. The HAL interconnect PCI card. *Lecture Notes in Computer Science*, 1362:15–29, 1998.

[3] Steven Lumetta, Alan Mainwaring, and David Culler. Multi-protocol active messages on a cluster of SMPs. In ACM, editor, *Proceedings of the ACM/IEEE Supercomputing Conference: November 15–21, 1997, San Jose, California, USA*. ACM Press and IEEE Computer Society Press, 1997.

[4] Ulrich Brüning and Lambert Schälicke. ATOLL: A high-performance communication device for parallel systems. In IEEE, editor, *Proceedings of the 1997 Conference on Advances in Parallel and Distributed Computing. March 19-21 1997, Shanghai, China.*, pages 228–234. IEEE Computer Society Press, 1997.

[5] Intel Corp. *Intel 450NX PCIset*, June 1998.

[6] Raoul A. F. Bhoedjang, Tim Rühl, and Henri E. Bal. User-level network interface protocols. *IEEE Computer*, 31(11):53–60, November 1998.

[7] Dave Dunning, Greg Regnier, Gary McAlpine, Don Cameron, Bill Shubert, Frank Berry, Anne Marie Merritt, Ed Gronke, and Chris Dodd. The Virtual Interface Architecture. *IEEE Micro*, 18(2):66–72, March/April 1998.