TCCluster: A Cluster Architecture Utilizing the Processor Host Interface as a Network Interconnect

Heiner Litz Maximilian Thuermer Ulrich Bruening

University of Heidelberg
Computer Architecture Group
Germany
{heiner.litz, maximilian.thuermer, ulrich.bruening}@ziti.uni-heidelberg.de

***Abstract*-So far, large computing clusters consisting of several thousand machines have been constructed by connecting nodes together using interconnect technologies as e.g. Ethernet, Infiniband or Myrinet. We propose an entirely new architecture called Tightly Coupled Cluster (TCCluster) that instead uses the native host interface of the processors as a direct network interconnect. This approach offers higher bandwidth and much lower communication latencies than the traditional approaches by virtually integrating the network interface adapter into the processor. Our technique neither applies any modifications to the processor nor requires any additional hardware. Instead, we use commodity off the shelf AMD processors and exploit the HyperTransport host interface as a cluster interconnect. Our approach is purely software based and does not require any additional hardware nor modifications to the existing processors. In this paper, we explain the addressing of nodes in such a cluster, the routing within such a system and the programming model that can be applied. We present a detailed description of the tasks that need to be addressed and provide a proof of concept implementation. For the evaluation of our technique a two node TCCluster prototype is presented. Therefore, the BIOS firmware, a custom Linux kernel and a small message library has been developed. We present microbenchmarks that show a sustained bandwidth of up to 2500 MB/s for messages as small as 64 Byte and a communication latency of 227 ns between two nodes outperforming other high performance networks by an order of magnitude.**

Keywords-Low latency, interconnect, high bandwidth, HPC, HyperTransport, AMD, Opteron

I. Introduction

In High Performance Computing (HPC) there still exist Grand Challenges that demand for more powerful, less expensive and less power consuming machines than currently available. Since Roadrunner [1] has broken the petascale barrier in 2008, the HPC industry and scientists from all over the world are competing to build the first exascale computer. Analyzing the last ten TOP500 lists [2] a trend for HPC platforms can be observed. They are moving from big symmetric multiprocessor (SMP) machines towards large clusters consisting of thousands of interconnected nodes. The main reasons for this trend are that clusters scale better and that they can use commodity off the shelf components (COTS) which provide a much better price/performance ratio than specifically developed SMPs.

Almost all clusters are x86 CPU based and utilize processor hardware from either Intel or AMD. A cluster consists of several nodes which are comprised of one or more processors, memory and input/output (I/O) devices. To form a cluster out of such nodes a network interconnect is required. The traditional technology is Ethernet which is more and more getting replaced by faster and more efficient interconnects like Infiniband and Myricom. While network interconnect technology has been improved significantly over the years in terms of latency and bandwidth, it still represents the bottleneck of a cluster for many applications. For example, one of the fastest currently available implementations, namely the ConnectX Infiniband adapters from Mellanox [10] achieve a bandwidth of 1.4 GB/s and an end-to-end latency of about 1.4 us. This is only a fraction of the amount that modern processors are able to deliver at their chip interface. In an SMP system, multiple processors are interconnected by their native host interface like HyperTransport (HT) in the case of AMD or Quick Path Interconnect (QPI) in the case of Intel. Such interfaces are one order of magnitude faster delivering bandwidths of 10 GB/s and access latencies of as low as 50ns [4].

In an SMP machine, threads can communicate very efficiently via shared memory. This requires a cache coherency mechanism [5] like MESI which guarantees data consistency in the system at all times. While such a coher-

ency model facilitates programmability of shared memory systems it dramatically limits their scalability. The coherency mechanism requires the exchange of cache state information between the processors, which limits the performance gain. The shared memory approach performs well for small scale systems of up to 8 or 16 nodes, however, as the coherency overhead grows with the number of nodes it cannot be applied to large clusters.

To overcome the disadvantages of the current approaches we have developed a fundamentally new cluster architecture which we refer to as Tightly Coupled Cluster (TCCluster). TCCluster uses the processor's host interface as an interconnection network leveraging its high bandwidth and low latency characteristics while providing scalability of up to thousands of nodes without cache coherency. The key idea we present in this paper is to divert the coherent host interface between the processors from its intended use by operating it in a non-coherent fashion.

The original use of non-coherent interfaces is to access I/O devices like southbridges or coprocessors. Our presented technique, however, enables the use of that interface as a cluster interconnect to circumvent the scalability limitation of coherent shared memory systems. Instead of being limited to a small number of nodes our approach allows to interconnect thousands of machines with the bandwidth and latency characteristics normally only an SMP can provide.

For the implementation of our ideas we chose the AMD Opteron processor over an Intel based system. The main advantage of AMD is that their processors use HyperTransport (HT) as their host interface which is an open protocol. Furthermore, AMD provides comprehensive publicly available processor documentation in the form of the BIOS and Kernel Developers Guide (BKDG). AMD also supports the coreboot project which has the goal of developing a generic open source BIOS firmware for x86 platforms.

II. Related Work

Research on high performance interconnects to enable scalable parallel computer systems has a long history in computer science. Most approaches apply the message passing programming model which uses explicit messages for inter process communication. One of the very first milestones in this area was the *Transputer* [6] developed in the 1980ies. It was specifically designed for parallel systems and combined the processor with four serial links on a single chip. The in-built network allowed for very efficient communication between multiple Transputers which could be interconnected to form a so called *computing farm*. In many aspects our approach is similar to the Transputer approach, however, we have applied it to modern x86 based systems which dramatically differ from the old Transputer design. Since then, processor builders like Intel and AMD have abandoned the idea of incorporating network interconnects directly into their processors, although this would offer a dramatic increase in performance.

In [7] Joerg and Henry analyze the benefits of a tightly coupled processor-network interface which is realized through additional processor registers. Although this approach performs well, in contrast to our technique, it requires a modification of the processor hardware. This is well known to be very different endeavour.

More recent approaches like Infinipath [8], Cray's XT3's seastar interconnect [9] and VELO [11] focus on optimizing the interface between the processor and the network interface. In all three approaches, AMD's processor host interface HyperTransport is used to reduce latency and to increase bandwidth. Although, HyperTransport allows for a direct connection between the network adapter and the processor without intermediate bridging, the three approaches still have a significantly higher latency than our approach which completely eliminates the additional latency introduced by the network hardware in traditional approaches. The network interface which currently (2010) can be regarded as the state-of the art as it offers very good performance is the ConnectX Infiniband adapter manufactured by Mellanox. It provides bandwidths of up to 40Gbit per link and a latency as low as 1.4 us [3].

In addition to the message passing based systems, a lot of research activities focus on scalable coherent shared memory based systems. Such systems offer a very good performance for small systems as they employ a much faster interconnect in the form of the host interface. Their biggest disadvantage, however, is their limited scalability. There exist various approaches which address this issue, like the virtual shared memory architecture by Li [12], or the scalable coherent interconnect by Gustavson [13]. More recent approaches, like Horus [14] and 3-Leaf [15], target modern x86 based systems. They enable large cache coherent non uniform memory architectures (ccNuma) by extending AMD's HyperTransport protocol. By applying a directory based coherency mechanism they can moderately increase the scalability to 32 nodes.

All approaches have been focused on either increasing the performance of the network interconnect or increasing the scalability of shared memory systems. Neither has been able to combine the best of the two worlds and to develop a truly scalable tightly coupled architecture. In order to overcome these shortcomings, we present the TCCluster mechanism which offers the same bandwidth and latency performance of a shared memory based system while scaling to much larger systems.

III. BACKGROUND

Most related work as well as our approach targets AMD instead of Intel processors which provide a significant advantage. AMD employs an open host interface protocol called HyperTransport, which is maintained by the HyperTransport consortium. Therefore, before presenting our approach, an analysis of the AMD Opteron architecture and the HyperTransport protocol will be provided. The current AMD processor family, named K10 [16][17], implements several components on a single chip. In addition to the processing elements itself called cores, a processor node comprises memory controllers, several levels of cache memory, up to four outgoing HyperTransport links and a crossbar that interconnects the different components.

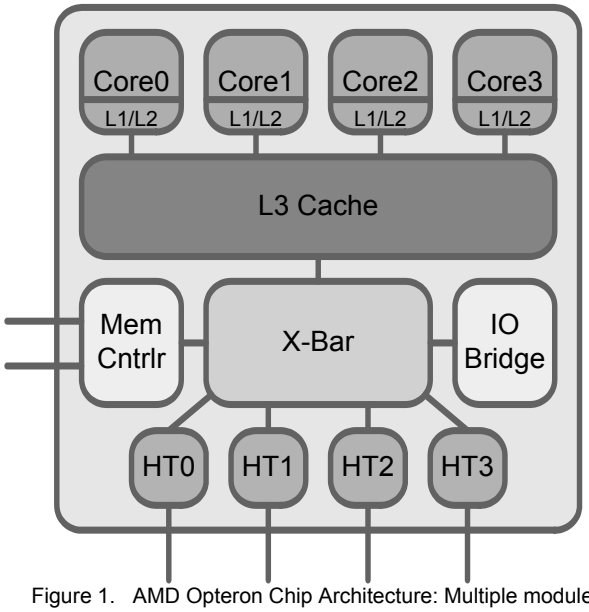

Figure 1. AMD Opteron Chip Architecture: Multiple modules including memory controllers and a crossbar switch are integrated on a single processor chip

Figure 1 shows the AMD Opteron chip architecture of a processor codenamed *Shanghai* that implements four cores that have their individual L1 and L2 caches and a large shared L3 cache that the cores can use to share data with each other. Furthermore, it contains an I/O bridge that converts between coherent and non-coherent HyperTransport packets, a DDR2 memory controller and four HyperTransport links. Those can be used to communicate with off-chip devices as other processors or as I/O devices.

HyperTransport is a packet based protocol that offers low latency (approximately 50 ns per hop) and a unidirectional bandwidth of 12.8 GByte/s per link. It has low overhead, uses multiple virtual channels for deadlock avoidance and defines fault tolerance mechanisms on the link level. HyperTransport links can be operated either in a coherent (cHT) or non-coherent (ncHT) fashion. Coherency is required for multi socket configurations which are available in two, four and eight socket configurations. The system shown in Figure 2 combines the physical memory modules which are connected to each processor to form a single shared memory address space. Such a system requires a mechanism like the MESI protocol to ensure cache consistency. The cache consistency guarantees for multiple cores that work on the same data values that each core has the same value in its cache at any point in time. Every time a data value is modified in a cache or loaded from main memory the other cores that participate in the coherent domain have to be informed and probed for a response. The transaction can only be completed if all nodes have responded to the probing. While this technique allows multiple nodes to a share a common memory address space, its scalability is limited. By increasing the number of nodes, the number of probe messages is increased proportionally which costs bandwidth and latency as the last incoming response pivotal. Furthermore, as the Opteron processor only contains four links, fully connected systems are only possible for two and four processor configurations. Larger systems have to utilize multihop topologies which increases the latency even further.

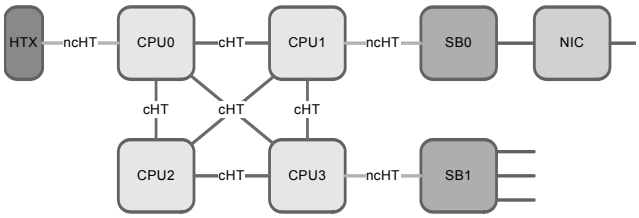

Figure 2. AMD Based Multiprocessor System: The processors nodes are interconnected by coherent HyperTransport links while the IO devices are attached over non-coherent links

To connect IO devices like southbridges or HyperTransport based accelerators [11] the links are operated in non-coherent mode. The non-coherent HyperTransport protocol defines a reduced set of commands which are used to transport data via Programmed I/O (PIO) or Direct Memory Access (DMA). A typical system that is comprised of multiple CPUs, two southbridges and a HyperTransport Extension (HTX) slot is shown in Figure 2. All four CPUs are directly connected via coherent HyperTransport links. In addition, the system features two southbridge chips that are connected to the CPUs via non-coherent links. These chips allow to attach PCI-Express, USB and SATA I/O devices to the system, for example network interface adapters. The HTX slot is a PCI like connector that can be used to attach ncHT devices [17] with very low latency directly to the CPUs without any intermediate bridging.

IV. Approach

AMD based multiprocessor systems are restricted to a maximum number of eight nodes. The reason for this restriction is the limited scalability of cache coherency protocols. To overcome this issue and to enable a much higher scalability, our approach abandons cache coherency. Multiple nodes are connected via a non-coherent interface that uses message passing for node to node communication. In contrast to other approaches like Ethernet or Infiniband, however, we rely on the system's host interface directly. As will be shown later, this provides a dramatic improvement in terms of bandwidth and latency over the existing approaches.

Connecting multiple nodes over a non-coherent interface which have their own independent address spaces is a common method to create large clusters or networks of computers. The traditional approach uses network interface cards (NIC) that offer various services to the host. The most important capability is the routing mechanism which ensures that a network packet is delivered to the correct node in the system. Furthermore, a NIC often provides DMA functionality to retrieve data from the sender and to deliver it into the receiver's main memory. Besides data packetization and sending the packet over the network, modern NICs often support fault tolerance mechanisms to ensure reliability and processing of packets in hardware by offloading certain tasks.

Unlike previous approaches our presented solution does not require any intermediate NIC hardware and, therefore, offers superior performance at a reduced cost. Connecting multiple processors directly with HyperTransport, however, raises the several challenges. First of all, the communication mechanisms that can be realized over an ncHT link have to be analyzed to develop a suitable programming model. Second, it has to be discussed how the links between two Opteron processors can be operated in a non-coherent fashion. Third, it has to be analyzed how packets can be sent over such an interface and how the routing of packets is performed. Fourth, it is necessary to discuss how the address mapping is done in such a system. Fifth, several issues regarding the operating system which is executed on the nodes have to be addressed and it has to be analyzed how the initial configuration, namely the boot sequence, can be performed. Sixth, the physical constraints of a possible implementation have to be defined to enable the development of a prototype.

It will be shown that a working proof-of-concept can be realized utilizing two Opteron nodes with a custom BIOS firmware, a modified Linux kernel and specific driver software. Both nodes are interconnected through a cable plugged into a HyperTransport Extension (HTX) connector that can be found on various server mainboards.

A. Programming Model

To develop a suitable programming model for TCCluster the capabilities of a non-coherent HyperTransport link have to be analyzed. HT defines three main types of message transactions: posted writes, non-posted reads and responses. While posted writes are completed as soon as they are sent, non-posted requests require some sort of book keeping. In HyperTransport read requests are realized using split phase transactions whereas the read requests are routed identically to write requests, however, the response to the request is not. Each read request creates an entry in the response matching table located in the northbridge and receives a tag. A matching response will carry the same tag and can be thereby routed without having to carry an address. The number of these tags is, however, limited and they are always mapped to a specific NodeID. This fact makes it impossible for our approach to route responses which means that the software can only communicate via writes and may not use read accesses. While this limitation sounds critical at first, it is not. It prohibits load/store communication, however, relying on remote stores is perfectly sufficient to implement the message passing or the global address space programming model. Also, as our approach cannot exploit additional hardware DMA, CPU offloading and interrupts are not supported.

Our approach supports both Message Passing as well as the PGAS programming model by utilizing remote stores. To support a Message Passing Interface (MPI) protocol like MVAPICH [18] an underlying application programming interface (API) is required that enables sending and receiving of messages. Within TCCluster, sending is performed by writing to a specific address that is mapped to a remote node. The message is delivered via the HyperTransport interface and written to a ring buffer in main memory at the target node. Receiving of messages is implemented by polling the corresponding address on the target node. As soon as the change in main memory is observed the API can extract the data from the buffer and copy it into main memory. It then has to overwrite the slot to free it for further use. Periodically, the APIs on the endpoints have to exchange pointer information to communicate buffer fill levels and to implement flow control. As there exists no hardware support for managing messages it is impossible to share receive buffer space between multiple endpoints. Therefore, each node has to allocate a 4 KB ring buffer for each endpoint it want to communicate with. While this limitation prohibits unlimited scalability the approach is sufficient to support hundreds of endpoints. Remote stores can also be utilized to implement one-sided rendezvous like communication. In this case data is written directly to the final destination on the remote node and an additional queue is used for synchronization and management.

TCCluster is compatible with PGAS implementations like UPC over GASNet. Whereas the data transfer (relaxed consistency operations) is straightforward, global synchronization messages implemented through remote stores are used to enforce strict sequential consistency. They can be realized through API managed software barriers. The HyperTransport fabric guarantees in-order delivery for packets within a single virtual channel throughout the network whereas the Sfence operation can be used as a serialization instruction to enforce sequential consistency in the processor. A detailed discussion of the software implementation on top of TCCluster is out of scope of this paper and we will focus on the mechanism itself in the following.

B. Non-coherent Configuration

HyperTransport links between two processors are generally configured as coherent. As soon as the Opteron processor emerges from its reset state it enters the low level initialization and begins to configure its HyperTransport links. Therefore, it drives some specific data patterns on the wires trying to detect another device that may reside on the other side of the link. If both endpoints conform to that sequence the link connection is established. Then, both endpoints identify themselves as a coherent or non-coherent device to determine the type of the link. In the case of two Opterons the link type will be coherent and the boot strap processor (BSP) can now use this link to configure itself and all other devices in the fabric.

TCCluster requires non-coherent links between the processors. While this mode is not intended nor referenced in any of the processor or HyperTransport specifications it is still possible to enforce such a behavior. The processors implement a specific register for debug purposes enabling non-coherent operation. This possibility is exploited by our approach. After the initialization phase the HyperTransport links are configured coherent which enables the BSP to access and set the debug register. The modifications become effective at the next warm reset which causes a re-initialization of the link, at which time, the processors identify themselves as non-coherent devices. This technique allows to enforce a non-coherent link between processors.

C. Routing

The Opteron northbridge performs different routing tasks depending on the source and the destination of a packet. If the packet was received from an IO link and targets main memory it is forwarded to the IO bridge which converts the non-coherent packet into a coherent packet. A coherent packet is either forwarded to the on-chip memory controller or to an outgoing HyperTransport link if the physical memory address resides on another node. Coherent packets that target an IO link are also forwarded to the IO bridge which converts it into a non-coherent packet and forwards it to the corresponding IO link. Non-coherent packets originating at an IO link that target another IO link are simply forwarded without bridging.

The routing process itself is split into two stages. The first step is to compare the address of every packet against the DRAM and MMIO address ranges which are defined by base/limit registers. This lookup returns the NodeID which defines the home node of the requested DRAM or I/O address. This NodeID then indexes the routing table which returns the corresponding HyperTransport link to which the packet should be forwarded. MMIO accesses which target an IO device that is connected to the local node are treated different. In this case the destination link is directly provided by the base/limit registers without the need of indexing the routing table. This fact is exploited by our approach which assigns NodeID zero to every node in the TCCluster and which maps every MMIO address range to NodeID zero as well. With this setup every northbridge believes to be the home node of any packet, thereby, directly forwarding it to the outgoing HyperTransport link.

D. Address Mapping

A regular Opteron based shared memory system consists of multiple processors with individual physical memory modules attached to each processor. The complete physical memory is aggregated to form a single shared physical address space and each processor has an identical memory map. In a TCCluster parts of the local physical memory are made accessible for remote nodes. This memory area can then be used to exchange messages between the nodes. In TCCluster local memory is treated as DRAM and remote memory is treated as MMIO space. This requires a different and unique address map for each node as depicted in Figure 3.

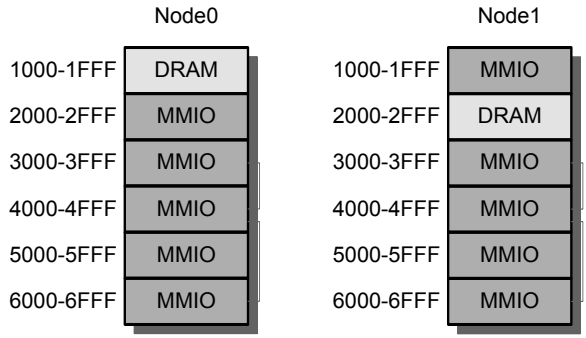

Figure 3. TCCluster Address Map: Both nodes have a different view of the TCCluster address space.

Write accesses from Node0 to the address range 1000-1FFF result in a local physical memory access. The same address range accessed by Node1, however, results in a network package that is transmitted over a non-coherent TCCluster link towards Node0. One can see that the address map presented in Figure 3 shows a contiguous global address space ranging from 1000 to 6FFF. A contiguous address space is necessary as the northbridge implements interval routing mechanism which can only map single contiguous address intervals to each outgoing HyperTransport link. Memory holes within a node specific address space are, therefore, impossible. If a system desires to provide only parts of the local memory to remote nodes, the driver has to restrict the address ranges that can be mapped into user space by remote nodes. Current Opteron processors support a physical address space of 48 bits. Therefore, the combined global address space in TCCLuster is currently limited to 256 Terabyte. Future AMD processors will increase the supported physical address space to 52 bit and beyond.

E. Initialization

Before computer systems are able to execute software and to run an operating system, they have to be initialized through BIOS firmware. Therefore, code that resides in non-volatile flash memory is executed by the processor to setup all devices in the system. In an AMD environment the code is retrieved via the southbridge which is connected to the BSP via a non-coherent HyperTransport link. In a multiprocessor setup one processor is assigned the Boot Strap Processor (BSP) which configures itself at first and then traverses its HyperTransport links to find the other Application Processors (APs) and I/O devices. Before the BSP is able to configure the routing tables in the processors it has to determine the topology of the system. This can be done either in a static way by passing the topology description to the firmware at compile time or dynamically at runtime. Therefore, the processor performs a depth-first search for all APs. After system reset each NodeID register in each AP is initially set to seven. If the NodeID register is still seven, the BSP knows that it hasn't visited that specific node yet, so it assigns a new NodeID to the AP and configures its routing table entries accordingly. The final topology then consists of a structure of all nodes and the HyperTransport links interconnecting them.

TCCluster introduces a new system type where multiple processors are interconnected via HyperTransport I/O links. In principle, each individual processor resides in its own coherent domain and, therefore, represents its own bootstrap processor. An individual southbridge for each processor is undesirable as it is costly and occupies a HyperTransport link which could be otherwise used as a network link. To approach this problem we introduce the notion of *Supernodes*. A *Supernode* consists of four or eight processors which are interconnected through coherent HyperTransport links and form a shared memory system. Figure 4 shows how multiple *Supernodes* can be connected via non-coherent TCCluster links to other Supernodes. A Supernode defines a single combined remote MMIO space which is spread over the different processors inside the *Supernode*. Therefore, a *Supernode* is addressed as a single entity making it completely transparent to the TCCluster.

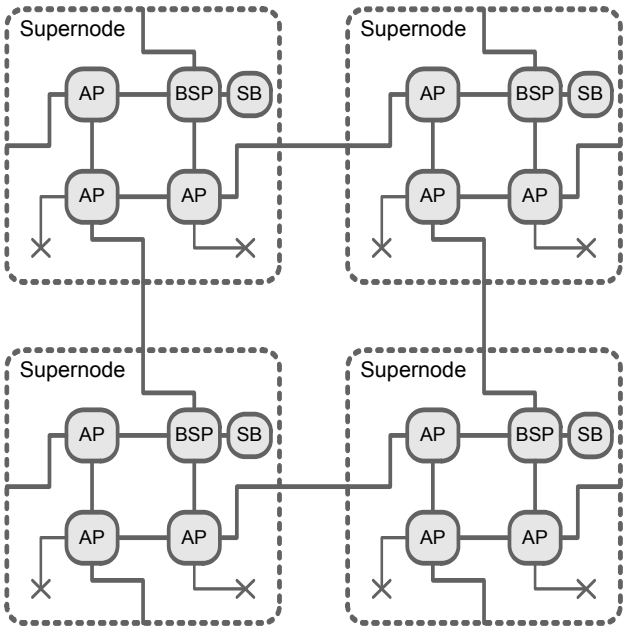

Figure 4. Four Interconnected Supernodes: Each *Supernode* contains a southbridge connected to the BSP which configures the other application processors.

Within such a system each *Supernode* is configured individually by its BSP. Therefore, each BSP needs a topology description and its rank within that topology to be able to configure the address map registers accordingly. In addition the Supernodes have to implement a common clock and a synchronized warm reset. A possible solution is to distribute the clock over each link and to use a PLL in each *Supernode* to build a truly synchronous system. This can be realized through a backplane which interconnects the various *Supernodes*.

F. Physical Implementation

Our proposed approach enables tightly coupled clusters from commodity off the shelf hardware. All main components including processor, memory and I/O are available, however, it is necessary to develop a proper mainboard and backplane which provides the processor interconnect. In contrast to traditional systems where nodes are connected via network cards, cabling and switches, TCCluster utilizes

the processor internal links directly. A possible TCCluster implementation therefore consists of several multi processor mainboards that act as *Supernodes* which are interconnected via TCCluster links through a backplane.

There exist two physical constraints that aggravate the design of such a backplane. First, AMD Opteron processors that communicate via HyperTransport require a mesochronous link clock that is derived from the same oscillator. Second, physical trace length of the links between two processors is limited to 24 inches.

A solution for the first issue is relatively simple. As the input clock of all processors has to be mesochronous (same frequency) but not synchronous (same frequency and phase) it is sufficient to employ a single clock for the complete system which is then fanned out to the different processors via clock distribution ICs. Jitter cleaners can be applied to compensate for the jitter that is introduced by the backplane traces.

To solve the second issue it is required to match the topology of TCCluster with the location of the *Supernodes* within the system. For an *n*x*n* mesh the distribution that minimizes trace length between the *Supernodes* would be to employ *n Supernodes* in the horizontal axis and *n Supernodes* in the vertical axis. A blade type rack server where the nodes are arranged vertically next to each other and then stacked in multiple rows provides a very balanced distribution of nodes in the *x* and *y* axis. Furthermore, the trace length limitation is specified for copper traces on FR4 PCB material. Coaxial copper cables can provide much better signal integrity and fewer resistive loss enabling longer trace lengths as defined by the specification.

V. PROOF-OF-CONCEPT

The discussion in the previous paragraphs motivates a real world implementation of TCCluster. However, designing a complete mainboard and backplane interconnect is an extremely challenging and cost intensive task. Thus, we opted for a proof-of-concept implementation which can prove the merit of our work. This approach allows us to focus on developing the required firmware and software for implementing TCCluster which is already a very complex task. In particular, we built two different prototype systems.

The first consists of a single Tyan S2912E mainboard which provides two processor sockets, both populated with AMD quad core Shanghai processors. The mainboard provides two HyperTransport links between processor Node0 and processors Node1 which can be aggregated to a dual link. A third HyperTransport link is routed from Node0 to the southbridge chip which interfaces to I/O and provides the BIOS firmware. A forth link is routed from Node1 to an HTX slot. Using this setup we configured one of the HT links between the processors as a TCCluster link and the other as a regular coherent HT link. The coherent link allowed us to access the Node1 from BIOS firmware which is required to modify the processor registers and to execute code. As a TCCluster link is write only the coherent link allowed us to check whether our approach actually works and whether we can successfully transfer data over the TCCluster link.

The second prototype consists of two Tyan S2912E motherboards interconnected via an external TCCluster link as shown in Figure 5. Therefore, we designed a cable which plugs into the HTX slots on both boards interconnecting them via a direct HT link. For this installation it is required to run the modified firmware on both machines and power them up simultaneously. This can be achieved by short-circuiting both reset and power up signals from the two machines. For both approaches the BIOS firmware we developed is nearly identical.

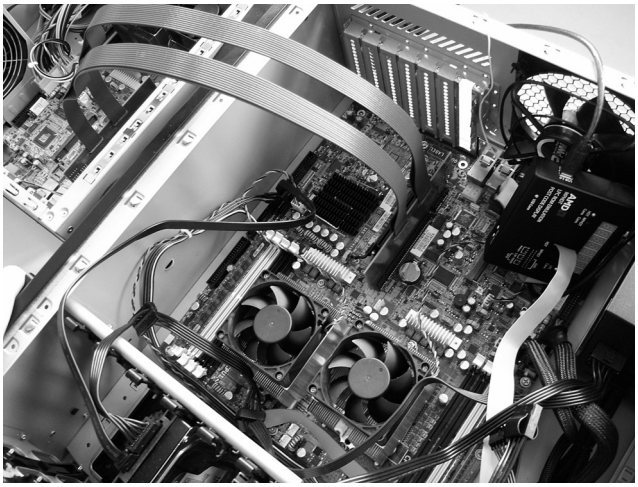

Figure 5. TCCluster Prototype consisting of two Tyan machines interconnected by our HTX cable adapter.

As a foundation for our code we use coreboot, also referred to as LinuxBIOS, which is an open source BIOS firmware project that already supports many AMD and Intel platforms. We first applied modifications to support the new Shanghai processors and then rewrote most part of the coherent and non-coherent link enumeration code to implement the following sequence which configures an HT link as a TCCluster link.

- Cold Reset: Both platforms come out of cold reset simultaneously and perform their HyperTransport low level link initialization. The TCCluster link gets configured as coherent.
- Coherent Enumeration: Both platforms perform the usual boot sequence including coherent link enumeration. At this point the TCCluster links are still configured as coherent which would cause the regular

firmware to perform a search for all coherent links thereby building the system topology. The modified TCCluster firmware avoids this by ignoring such links and only performs coherent link enumeration for the nodes within a *Supernode*.

- Force Non-Coherent: Each TCCluster link in the system is forced into non-coherent mode. Furthermore, the link speed is increased from 400 to 4.800 Mbit/s.
- Warm Reset: Both platforms or nodes issue a warm reset, which results in another low level link initialization. The modified settings now become effective and the TCCluster link gets configured as non-coherent.
- Northbridge Init: Both platforms configure their northbridge including nodeID, DRAM address range, MMIO address range registers and routing registers as described in the previous paragraphs. For the first prototype, reconfiguration is performed via the coherent link between Node0 and Node1, in the second setup each machine configures itself individually.
- CPU MSR Init: The Memory Type Range Registers (MTRR) on both nodes are reconfigured to map a large uncachable address space to the TCCluster MMIO link. This causes the processor's system request queue to generate non-coherent posted HT packets which are required for TCCluster.
- Memory Init: The machines initialize their memory controllers and report the size and type of memory which is attached to the processors.
- EXIT CAR: Until now the firmware is executed in cache as RAM (CAR) mode which means that the code is fetched from the firmware ROM and the L3 cache is treated as memory. At this point the system is comparatively slow as the performance is limited by the read bandwidth of the ROM. To exit CAR, the firmware is copied into main memory and the program counter gets pointed to main memory.
- Non-Coherent Enumeration: The processors interconnected by a TCCluster link appear as non-coherent devices which causes regular firmware to perform I/O device enumeration for this link. This needs to be disabled for each TCCluster link.
- Post Initialization: The firmware performs TCCluster independent tasks and loads the operating system
- Loading Operating System: After the firmware configuration is completed the operating system, in our case Linux, can be loaded. The OS also switches the system from 32 bit protected mode into 64 bit user mode.
- Enabling Remote Access: The device driver maps the remote address range as memory mapped IO and provides access to the API.
- Data Transmission: The API requests page wise memory mapping of remote addresses into user space. User software can now access remote memory.
- Data Reception: On the remote node the same physical memory address, which is DRAM based in this case has to be mapped into user space which allows receiving data from remote nodes.

VI. Evaluation

To evaluate the TCCluster interconnect we provide software microbenchmarks that show the latency and bandwidth performance of our technique. We developed a Linux driver which can map remote TCCluster memory addresses into the user space and a rudimentary message library which can be used to send and receive messages. As the operating system we run Linux with a custom 2.6.34 kernel. We needed to compile our own Kernel to comply with a limitation of TCCluster caused by interrupts. Within the HyperTransport fabric interrupts are broadcasted to inform coherent and non-coherent devices within the system about specific events. It is required to avoid broadcasting of interrupts over TCCluster as interrupts have to be handled within the system and must not be send over the network. Therefore, all system management calls (SMC) need to be disabled which can be only achieved with a custom kernel.

The user space message library provides the following functionality. It can open local and remote memory addresses by calling the TCCluster device driver. The send function can then be used to transfer data to remote memory addresses. The receive function is called on the remote side to receive data from local memory. TCCluster transactions cannot generate cache invalidation requests on the receiver side. Therefore, the receiver needs to map the local memory which is accessible by the remote nodes as uncachable. This guarantees that all reads to remote node accessible memory bypass the cache and directly target main memory. Although, this approach generates additional processor-memory bus overhead when polling the memory it is the only way to guarantee that incoming data is seen by the processor.

The message library will offer support for synchronization primitives using the Sfence machine instruction. Sfence enforces a strict ordering between store transactions which can be utilized by the library to implement a synchronization mechanism. The message library provides the basis for higher level middleware. In principle, partitioned global address space (PGAS) implementations like GASNet or message passing protocols like MPI can be realized on top of our library, although, their support is out of scope of this paper.

In the following we present the latency and bandwidth performance of a two node system interconnected by a single bidirectional TCCluster link as shown in Figure 5. We used four Shanghai quadcore Opterons with 4 MB L3

cache running at 2.8 GHz and equipped with 8 GByte of main memory per node. Although, the processors, support 16 bit wide links with up to 5.2 Gbit/s per lane, due to signal integrity issues of our cable based approach we support only frequencies of up to 1.6 Gbit/s per lane. Future implementations that offer better cabling or routing the TCCluster links over a backplane will support higher frequencies and increased performance. Our approach makes intensive use of the write combining capability to generate maximum sized HyperTransport packets which reduce the command overhead. Therefore, multiple 64 bit store instructions are collected in the write combining buffer and sent out as a single packet.

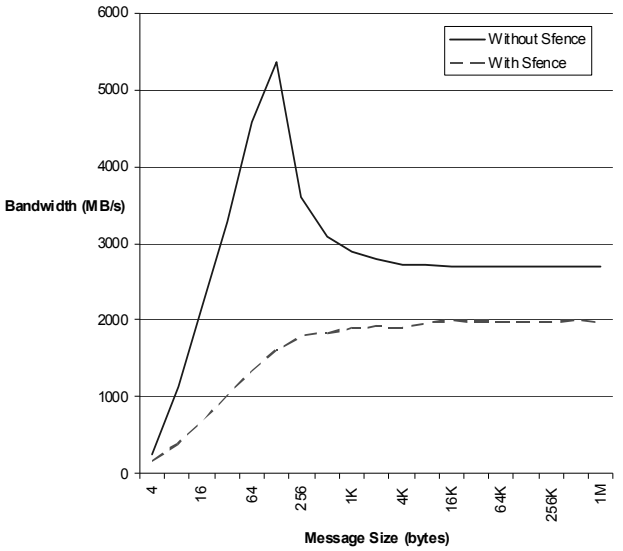

Figure 6. TCCluster Bandwidth

Figure 6 shows the bandwidth of a 16 bit wide TCCLuster link running at HT800 which equals 1.6 Gbit/s per lane. Our message library supports two different send mechanisms which are both shown in the graph. The first mechanism guarantees strictly ordered data transport by flushing the write combining buffers immediately. Therefore, after each cache line sized store operation an Sfence instruction is triggered. Sfence performs a serializing operation on all store instructions that were issued prior the Sfence instruction which introduces overhead limiting the write performance to 2000 MB/s. Higher bandwidth can be achieved with weakly ordered writes. In this case no fence operation is issued and the write combining buffers are flushed automatically in the case of a buffer overflow. In this case TCCluster provides a sustained bandwidth of 2700 MB/s. The peak bandwidth of 5300 MB/s that can be observed at 256K leverages caching structures within the Opteron and does not reflect the bandwidth performance of the TCCluster link. The Opteron provides eight write combining buffers which support a very high data rate. As we trigger no Sfence instructions the store queue can fill all write combining buffers before the outgoing TCCluster link becomes the bandwidth bottleneck. Weakly ordered writes as provided by the second mechanism are sufficient for most programming models as long as a serialization function as Sfence exists. For many data transfers ordered delivery of the message segments is not required as long as the a synchronization operation exists that can finalize the transaction.

As a baseline, the Infiniband ConnectX network adapter from Mellanox can be referenced [10]. It provides an MPI bandwidth of 2500 MB/s for 1 MB messages, 1500 MB/s for 1K messages and 200 MB/s for cacheline sized messages. Although, our evaluation does not include the overhead of the MPI middleware it can be seen that TCCluster provides a significant performance edge over Infiniband especially for small messages.

In the second microbenchmark we measured the communication latency that can be achieved with our approach. We used a standard ping pong kernel whereas the receive node polls a specific memory location and sends back a response as soon as the first message arrives. As shown in Figure 7, TCCluster provides a very low half-round-trip latency for 64 byte packets between two nodes of 227 ns. Even for 1 KByte messages the latency is still below 1 us. Other high performance networks like Infiniband currently achieve end-to-end latencies of around 1 us for minimal sized packets which leads to a 4X performance advantage for TCCluster. We also measured multi-hop latencies by binding the benchmark process to different processor sockets using numactl and comparing the results. It could be observed that each hop increases the end-to-end latency by less then 50 ns. This low latency guarantees scalability of TCCluster as networks consisting of many nodes can still communicate with low end-to-end latency.

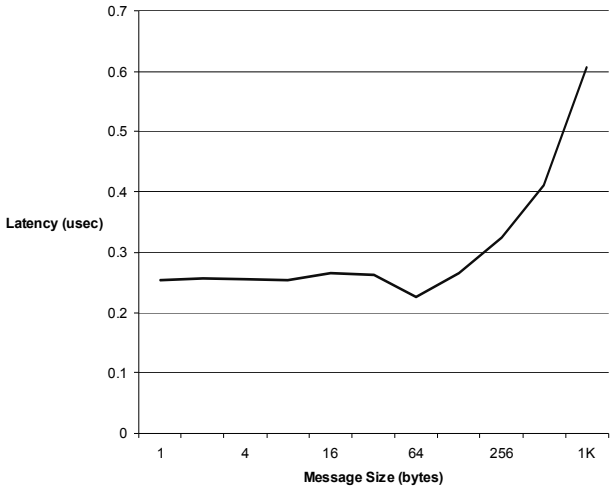

Figure 7. TCCluster Latency

VII. Conclusion and Outlook

We presented a novel technique named TCCluster for interconnecting large cluster systems by utilizing the processor host interface as a direct network interconnect. By virtually placing the network interface into the processor we achieve much higher bandwidths and lower latencies than traditional interconnects as Ethernet and Infiniband. Our approach does not require any hardware modifications and is realized entirely through firmware and software modifications. In particular, the HyperTransport host interface implemented in any AMD Opteron processor is exploited and reconfigured as a network interface. Our solution achieves an outstanding software-2-software network latency of 227 ns and a bandwidth of more than 2500 MByte/s for 64 Byte messages.

The next step in our work will be to port a middleware software layer like MPI or GASNet on top of our simple message library. This will enable to run more complex applications on the TCCluster system and to benchmark their performance.

Acknowledgement

We want to thank Advanced Micro Devices (AMD) for their donations and Parag Beeraka from AMD for the continuos support.

References

[1] Barker, K. J., Davis, K., Hoisie, A., Kerbyson, D. J., Lang, M., Pakin, S., and Sancho, J. C. 2008. Entering the petaflop era: the architecture and performance of Roadrunner. In Proceedings of the 2008 ACM/IEEE Conference on Supercomputing - Volume 00 (Austin, Texas, November 15 - 21, 2008). Conference on High Performance Networking and Computing. IEEE Press, Piscataway, NJ, 1-11.

[2] TOP500 Supercomputer List –Dongarra, Meuer, et al. June 2009 URL: http://www.top500.org

[3] Sur, S. Koop, M.J. Lei Chai Panda, D.K.: Performance Analysis and Evaluation of Mellanox ConnectX InfiniBand Architecture with Multi-Core Platforms. Proc. of 15th. Annual IEEE Symposium on High-Performance Interconnects, 2007. HOTI.

[4] HyperTransport Consortium: HyperTransport I/O Link Specification, revision 3.10, 2008.

[5] James Archibald, Jean-Loup Baer: Cache coherence protocols: evaluation using a multiprocessor simulation model. In Proceedings of ACM Transactions on Computer Systems (TOCS), 1986.

[6] Colin Whitby-Stevens. The Transputer. In Proceedings of the 12th ISCA, 1985.

[7] Henry DS, Joerg CF. A Tightly-Coupled Processor-Network Interface. In Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS V), 1992 .

[8] Dickman, L. Lindahl, G. Olson, D. Rubin, J. Broughton, J.: Pathscale InfiniPath: a first look. In Proceedings of 13th Symposium on High Performance Interconnects, 2005.

[9] Ron Brightwell, Kevin Pedretti, Keith D. Underwood: Initial Performance Evaluation of the Cray SeaStar Interconnect. In Proceedings of 13th Symposium on High Performance Interconnects (HOTI'05), 2005.

[10] Sayantan Sur, Matthew J. Koop, Lei, Dhabaleswar K. Panda: Performance Analysis and Evaluation of Mellanox ConnectX InfiniBand Architecture with Multi-Core Platforms. In Proceedings of the 15th Annual IEEE Symposium on High-Performance Interconnects, 2007.

[11] Heiner Litz, Holger Froening, Mondrian Nuessle, Ulrich Bruening: VELO: A Novel Communication Engine for Ultra-Low Latency Message Transfers. In Proceedings of the 2008 37th International Conference on Parallel Processing (ICPP), 2008.

[12] Li K, Hudak P. Memory coherence in shared virtual memory systems. ACM Transactions on Computer Systems (TOCS). 1989.

[13] Gustavson D. The Scalable Coherent Interface and related standards projects. IEEE Micro. 1992.

[14] Rajesh Kota, Rich Oehler: Horus: Large-Scale Symmetric Multiprocessing for Opteron Systems. In Proceedings of IEEE Micro, vol. 25, pages 30-40, 2005.

[15] 3 LEAF SYSTEMS: Enabling the Dynamic Data Center, whitepaper, 2009.

[16] P. Conway, B. Hughes: The AMD Opteron Northbridge Architecture, IEEE Micro, 2007.

[17] AMD Microsystems: BIOS and Kernel Developer's Guide (BKDG) For AMD Familiy 10h Processors, rev. 3.00.

[18] MVAPICH: MPI over InfiniBand and iWARP. http://mvapich.cse.ohio-state.edu/.