UNIVERSIDAD POLITECNICA DE VALENCIA

A Dynamic, Partitioned Global Address Space Model for High Performance Clusters

S. Yalamanchili and J. Young

School of Electrical and Computer Engineering
Georgia Institute of Technology, USA

J. Duato and F. Silla

Universidad Politecnica de Valencia,
Spain,

Technical Report: GIT-CERCS-08-01

Abstract

Memory-to-memory latency is a critical performance determinant of scalable computing systems. The use of modern interconnect fabrics tightly coupled to the processor-memory hierarchy such as AMD's HyperTransport™ (HT) have the potential to provide the lowest end-to-end transfer latency for systems comprised of tens to thousands of multicore nodes. However, to productively harness this raw capability, it must be exercised in the context of a global system model that defines how the system wide address space is deployed and utilized. Towards this end we advocate and explore the implications and implementation of a Partitioned Global Address Space (PGAS) model for the implementation of scalable cluster systems. A prototype implementation based on HT-Over-Ethernet (HToE) is proposed that is suitable for experimentation and measurement. In particular, we are concerned about the portability of the model and software implementations across future generations of processors with increasing physical address ranges. The paper concludes with the identification of several potential directions for future research.

1 Introduction

The purpose of this paper is to describe a model and propose a set of mechanisms for the realization of a partitioned global address space (PGAS) using commodity interconnects and outline an experimental implementation for HyperTransport based computing systems. The latter is intended to serve as a basis for application driven experimentation and measurement for PGAS systems. In such systems, physically addressable memory in all nodes is part of a global address space with non-uniform access time from any specific node. From the perspective of a node, the global address space is comprised of *local partitions* and *remote partitions* where the former can be accessed with the lowest latency and the latter can be accessed with larger and possibly non-uniform latencies (across distinct remote partitions). In our model, a local partition refers to DRAM accessed through a tightly integrated memory controller (MC). Remote partitions are accessed via read/write transactions or `get`/`put` operations that are implemented via message passing. The PGAS memory systems are an active area of research in the high performance and high productivity computing arenas, particularly within the language and optimizing compiler communities. The PGAS model can support both shared memory and message passing communication models. The simple semantics of get/put operations enables effective reasoning about performance and correctness and thus leads to optimizations and opportunities for performance and productivity enhancements that would otherwise be infeasible. Note that we are focused on the lowest latency accesses across a globally *non-coherent* shared physical address space.

We anticipate designs that should scale from 10s to 1000s of compute nodes. This end-to-end latency is currently dominated by software (particularly for message passing) and *intra*-node delays (wire ↔ memory) at the end points of communication. Our strategy is to focus first on a minimal set of mechanisms and abstractions to construct large scale systems capable of *minimal* latency implementations for data transfers between remote memory partitions, i.e., an implementation of the `get`/`put` model. Shared memory and message passing abstractions will be layered on this basic functionality. Finally, our approach rests on the presumption of the eventuality of 64-bit physical address spaces. Thus, the near term realizations proposed here are

viewed as emulations of 64-bit address spaces using addressability of current cores and is approached with the goal of ensuring software portability across generations of machines with the increasing physical addressability of future cores.

The remainder of this document describes an extension to the PGAS model - Dynamic PGAS or D-PGAS - and its realization for systems interconnected with Ethernet and using HyperTransport at the end nodes for reducing the (wire ↔ memory) latency. This model is distinguished in a few ways from the traditional high performance computing (HPC) version of the PGAS model. This paper concludes with some thoughts and recommendations for novel, high-value research directions enabled by this model.

2 The Dynamic PGAS Model

This section describes extensions to a well-known PGAS model [13] to permit flexible, dynamic management of a physical address space. The two components of this model are the architecture model and the memory model.

2.1 Architecture Model

Future high-end systems are anticipated to comprise of multi-core processors that access a distributed global 64-bit physical address space. Cores nominally have dedicated L1 caches for instructions and data, but may share additional levels of cache amongst themselves in groups of 2 cores, 4 cores, etc. A set of cores on a chip will share one or more memory controllers and low latency link interfaces integrated onto the die. An example of the latter includes AMD's HyperTransport™ (HT) protocol [9]. All of the cores also will share access to a memory management function that will examine a physical address and route this request (read or write) to the correct memory controller (MC) – either local or remote. For example, in the current generation Opteron systems such a memory management function resides in the System Request Interface (SRI) which is integrated on chip with the Northbridge [14]. Several such multicore chips can be directly connected via point to point links. This is the configuration made feasible by AMD's Opteron series multi-core processors leading to 2, 4 and 8 socket configurations with low latency access across 2, 4, and 8 nodes via direct HT connections.

Alternatively, the remote memory controller may not be directly accessible over a few HT links, but rather may be accessible through a switched network such as Infiniband [10] or a custom interconnect such as employed in high-end computing configurations by Cray [3]. In this case a `get` or `put` operation must be encapsulated into a message and transmitted to be serviced by the remote MC that will subsequently generate a response to the local MC. In this model memory controllers receive `put` and `get` transactions from any core. Finally, we believe the lowest latency is achieved when the MC is tightly integrated with the Network Interface (NI) minimizing the distance from the DRAM to the wire. We explore some of the options for such integration, although the prototype we are constructing cannot implement such a feature, since the MCs and NIs are integrated on-chip. However, we are constructing independent microarchitecture models of such integrated components.

The architecture model is memory-centric in the following sense. Cores are becoming primitive architectural elements that are no longer the primary determinant of performance because clock

frequency is bounded by heat dissipation and effective instruction issue width is bounded by control and data dependencies. Thus, computation scaling will come from the availability of additional cores and thread-level and data-level parallelism. Power dissipation concerns will accelerate the move to simpler streamlined cores, little or no speculation, and doubling of cores across technology generations. Memory bandwidth and interconnection bandwidth will have to track the increase in the number of cores and thus they will need to be effectively utilized to sustain Moore's Law performance growth with the scaling of cores. *Consequently, we are focused on the distribution of memory controllers in the system and their interaction with the interconnection network, which must deliver the lowest latency and highest bandwidth.*

2.2 Memory Model

The memory model is that of a 64-bit partitioned global physical address space. Each partition corresponds to a contiguous physical memory region controlled by a single memory controller and in this document all partitions are assumed to be of the same size. For example, in the Opteron (prior to Barcelona core) partitions are 1 TByte corresponding to the 40-bit Opteron physical address. Thus, a system can have 2^{24} partitions with a physical address space of 2^{40} bytes for each partition. Although large local partitions would be desirable for many applications such as databases, as we will point to in Section 4, there are non-intuitive tradeoffs between partition size, network diameter, and end-to-end latency that may motivate smaller partitions. Further, smaller partitions may occur due to packaging constraints, for example, the amount of memory attached to an FPGA or GPU accelerator via a single MC is typically far less than 1 Tbyte. Thus we view the system as a network of memory controllers accessed from cores, accelerators, and I/O devices.

We need to distinguish between two classes of memory operations generated by a local core - i) load/store operations that are issued by cores to their local partition and are serviced as per specified core-semantics, and ii) `get`/`put` operations that correspond to one-sided read/write operations on memory locations in remote partitions. The `get`/`put` operations are native to the hardware in the same sense as load/store operations. The execution of a `get` operation will trigger a read transaction on a remote partition and the transfer of data to a location in the local partition, while the execution of a `put` operation will trigger a write of local data to a remote partition. Transactions may have posted or non-posted semantics. The `get`/`put` operations are typically visible to and optimized by the compiler. The address space is non-coherent, although we discuss the layering of on-demand coherence on this D-PGAS model in Section 4.2. We intentionally separate the issue of coherence from the issues central to realizing a low latency PGAS model in the interests of opportunities to develop scalable coherence techniques. Now consider a `get` transaction on a memory location in a remote partition. This read transaction must be forwarded over some sort of network to the target memory controller and a read response is transmitted back over the same network. The specific network is not germane to the D-PGAS model implementation. However the fact that NIs are tightly integrated with the MCs is relevant. Being constrained by commodity parts, our prototype utilizes HT-over-Ethernet (HToE).

An application (or process) now is allocated a physical address space that may span multiple partitions, i.e., local and remote partitions. Equivalently, the processes' physical address space is mapped to multiple MCs, and thus an application's virtual address space may map to physical

pages distributed across local and remote partitions. There are a number of reasons to physically distribute an application's physical address space. The most intuitive one is for sharing where processes from an application may share physical pages by having portions of their virtual address spaces mapped to the same physical pages. No assumptions need to be made about how that shared space is managed, leaving open options for optimized management that are specific to the type of shared interactions, e.g., communication and shared libraries. There is a significant body of knowledge on managing remote storage that can be brought to bear in this context. One of our principal research directions is to explore how such flexible mappings can be used to optimize all aspects of memory system behavior including coherence, consistency and sharing.

The set of physical pages allocated to a process can be static (compile-time) or dynamic (run-time). The nature of the page management changes in scalable systems due to the hierarchy of latencies necessitating optimizations that have little relevance in traditional operating systems, e.g., page placement. Physical partition and page allocation can affect the communication support that must be provided. For example, it may be necessary to maintain a list of remote partitions that can be accessed or information related to coherence/consistency management that may be maintained on a per page basis (more in Section 4.)

To summarize, our model specifies *get*/*put* transactions for accessing physically distributed pages of a process and these transactions may have posted or non-posted semantics. The location of physical pages may be changed under compiler or operating system control, but the pages remain in a global 64-bit physical address space. All remote transactions are necessarily split phase. We now turn our attention to the process of implementing an experimental D-PGAS system.

3 Implementation of a D-PGAS Prototype

To evaluate the new functional capabilities afforded by the model we are constructing a prototype for experimentation and measurement. The prototype defines how individual physical address spaces of each core are mapped into the global physical address space and the tunneling of associated *get*/*put* transactions through Ethernet.

3.1 Implementing Global Addresses

Our plan is to implement a test bed using AMD Opteron processors coupled via HT/Ethernet with PGAS support implemented in an FPGA-based HT-Ethernet bridge. While the model is based on a 64-bit flat address space, the AMD x86-64 architecture supports a 52-bit physical address and each individual Opteron core generates a 40-bit address while the Barcelona core will generate 48-bit addresses. We can expect the physical address range to continue to change in future generation cores, and consequently our solutions should anticipate such changes. Therefore, we implement mechanisms today that effectively can emulate a 64-bit address space. The physical address space of a process is mapped across the MCs in the system, and memory accesses must be directed to the correct MC. Accesses to local MCs, i.e., local partitions, are handled as they are today, for example in the Opteron. Remote accesses are handled by mapping the address to a remote MC. We propose to implement this mapping by adapting simple well-known architectural techniques for extending address ranges.

Recall that the memory management function (see Section 2.1) determines the MC that is to service the request. If it is a remote MC, the request is sent to the HToE bridge rather than the local memory controller. The basic principle is to select some number of upper bits of the physical address generated by a core to index a number of partition registers. The contents of the indexed partition register are concatenated with the remaining bits of the physical address to produce a system wide 64-bit address. The upper address bits for this 64-bit address can be used to route the read or write transaction to the correct remote MC. For example, in the case of the Opteron that produces a 40-bit physical address, the upper two bits can be used to index one of four 26-bit partition registers. The concatenation of the two addresses will produce a 64-bit address. By changing the contents of the partition registers, a core can address system wide memory. In our prototype, the HT-to-Ethernet (HToE) bridge will perform this function as illustrated in Figure 1 (from [6]).

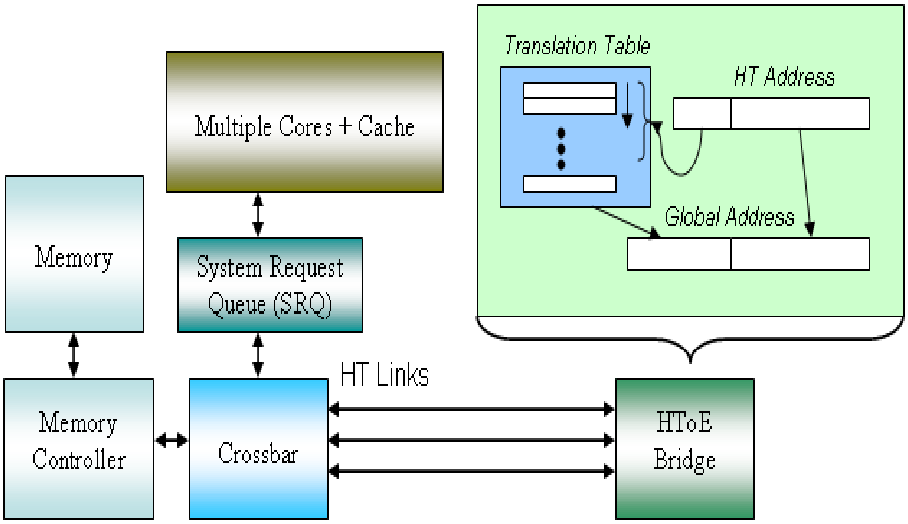

Figure 1: Logical Implementation of D-PGAS Address Translation

This preceding implementation will allow two nodes to access the same remote partition- the partition whose address is stored in the individual partition registers of their individual translation tables. Alternatively, the shared partition may be local to one node (i.e., does not traverse HToE) while it is remote to the other node (i.e., traverses HToE). If partition registers are not modified over time, i.e., statically assigned at initialization, communication between two processes requires that i) their physical address spaces overlap, or ii) they exchange memory regions for data transfers as is done in modern RDMA systems and then use this information to update their partition registers. If partition registers are modified over time, different areas can be used for communication. In the extreme case where a partition register is modified prior to each remote access, we have the equivalent to message passing (the partition register is effectively set with the destination node ID) with remote DMA capability.

This model provides a great deal of flexibility. For instance, a given process may have access to multiple remote partitions at a given instant in time. If certain partitions are only used by a host core, these partitions become private memory not accessible by other cores. If certain partitions are accessible from different cores, this enables communication among them through shared memory. If a given partition is accessible to only two cores, this enables private communication

among that pair of cores. Finally, the ability to dynamically change the value of the mapping means that each core can get access to a range of addresses (over time) larger than the range defined by 40-bit addresses. This dynamic address mapping could prove useful in implementing a flexible hardware virtualization scheme.

Once the system is initialized, global addressability is used for creating, possibly dynamically, shared memory partitions or message passing partitions. The latter can implement traditional push messaging or alternatively a pull message model. The overhead of message passing can be significantly reduced. Specifically, for pair wise communication, we anticipate that the overhead of remote DMA or RDMA can be significantly reduced. Finally, with 48-bit addressing providing global physical address spaces of 256 TBytes, increasing physical address space may initially be viewed as unnecessary. However, the techniques proposed here are useful for i) implementing PGAS models for cores that provide both 32-bit and 40-bit addresses, ii) for extending these address spaces across a variety of interconnection fabrics, iii) hosting a variety of optimizations concerning inter-cluster communication, iv) implementing sharing between clusters across the network, v) enabling more flexible partitioning and provisioning of resources in large scale compute and data server configurations, and vi) managing special purpose devices and compute accelerators. Further, it is possible to support a very large number of nodes with each potentially accessing a large (on the order of terabytes) physical memory. For example, note the products offered by Violin Memory Inc. [8].

This integration of memory and communication has several consequences. First, it will move communication from the I/O space to the memory space. By virtue of having communication reside in the memory space, we believe that the traditional high overhead of mediated NI access mechanisms can be removed from the latency critical path. Logically and physically, the interconnection network is now viewed as interconnecting memory controllers across the system rather than interconnecting cores via I/O subsystems. Communication services supported via I/O devices require mediation (operating system/virtual machine monitor (VMM)). Such mediation is a high latency operation, and a substantial body of work has evolved in amortizing, reducing, and hiding network interface (NI) latency. We also believe this integration enables new capabilities for cluster-based systems including fine-grained remote direct memory access (RDMA), page-based memory semantics, and simple reliability models for service provisioning in data centers (some thoughts in Section 4). Second, the tightly integrated network-memory system supports flexible provisioning of resources across nodes in support of virtualization. For example, a virtual machine (VM) executing on one node can have its physical memory space span multiple nodes. Finally, we anticipate that a system-wide name space simplifies service management and deployment, service redundancy and fault tolerance, service isolation for security purposes, and sharing in large data centers.

3.2 The HyperTransport-over-Ethernet Bridge

It is useful to first provide a simple overview of key operational attributes of HyperTransport. In its simplest instantiation, HT devices are connected via a point-to-point, one-dimensional topology anchored at one end by the host bridge. The host bridge implements the interface to the rest of the system. Data and control packets are transmitted over three classes of virtual channels - posted, non-posted, and response. HT device request packets travel upstream to a host bridge where they are either a) routed upstream to a higher level device or main memory b) routed back

downstream to the target device. As an example, a HT read request command packet with 40-bit addressing is shown in Figure 2.

The fields of this command packet are used to specify options for the read transaction, and to preserve ordering and deadlock freedom. The most important fields for our example are the UnitID, SrcTag, SeqID and address. The UnitID specifies the source or destination device and allows the local host bridge to direct requests/responses. The SrcTag and SeqID are used to specify ordering constraints between requests from a device, for example ordering between outstanding, distinct transactions. Finally, the address field is used to access memory that is mapped to either main memory or HT connected devices. As shown in the next section, an extended HT packet builds on this format to specify 64-bit addresses [9].

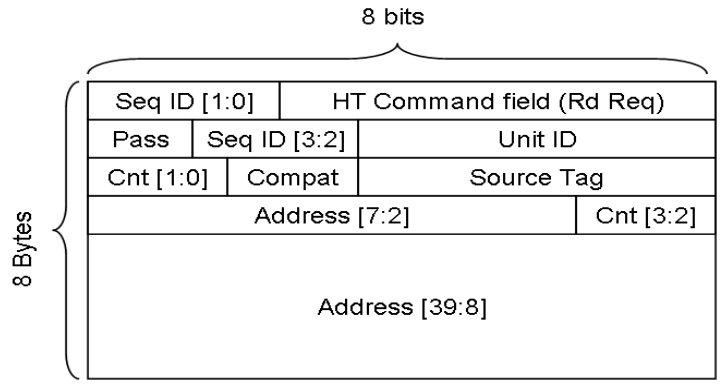

Figure 2: HT Request Packet

The HToE bridge implementation uses the University of Mannheim's HyperTransport Verilog implementation [7]. We retain their application interface to communicate between the local HT link and the HToE bridge.

3.2.1 Address Translation and Ethernet Encapsulation

As a practical matter, our implementation will be based on Opteron nodes where each Opteron node will have an Ethernet-enabled FPGA card available in the HTX connector slot. Several nodes will be connected via an inexpensive Ethernet switch. Depending on the specific board and FPGA implementations, the HT FPGA may be available in the I/O or memory address space. This impacts efficiency but not the functionality we wish to test. The functionality has been implemented in synthesizable Verilog and will be integrated into the Opteron-based test bed. We may be able to access the Northbridge address mapping tables (via the BIOS perhaps) and specify the physical address space mappings. However, if not, and if the FPGA card is in the I/O space, this document assumes that the MMU will distinguish between accesses to the local memory and the I/O address space (or some mechanism will) and HT packets will be sent for non-local addresses through a HT-Ethernet bridge.

Consider a system that has been properly initialized, i.e., all of the partition registers in the D-PGAS bridges have been loaded for the application. Now consider a parallel, shared memory application that generates a read operation to an address that is in a remote partition. There are three stages in each individual communication operation (e.g. a read request command) at a

given source host and attached devices: a) extension from the 40-bit physical address in the Opteron to the 64-bit physical addresses, b) creation of a HyperTransport packet which includes a 64-bit extended address and c) mapping the most significant 24 bits in the destination address to a 48-bit MAC address and encapsulation into an Ethernet frame. Obviously, implementations can pipeline the stages to minimize latency, but retaining the three stages has the following advantages: i) it separates the issues due to current processor core addressing limitations from the rest of the system, which will offer a clean, global shared address space, thus allowing implementations with other true 64-bit processors, and ii) it will be easy to port to other platforms that do not encapsulate by using Ethernet frames, but use other link layer formats, for example, Infiniband.

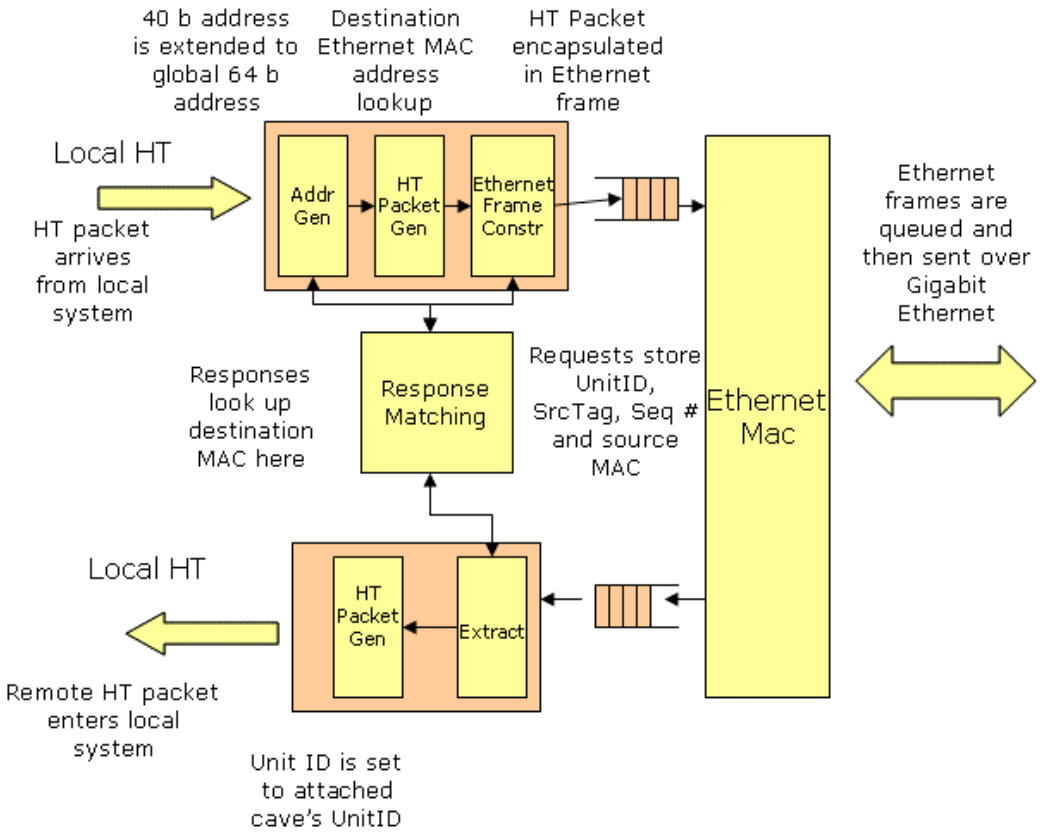

Figure 3: The D-PGAS Bridge with Read Request

The detailed transmit behavior of the HToE Bridge for a read request to a remote partition is described by Figure 3. Note that this figure shows a single end point. In the description below the transmit side of the request and the receive side of the request takes place at two physically distinct end points. We use the same figure to describe the processing of the message at the source node and destination node since their designs are identical to that shown in the figure.

First, the HT packet type is decoded into a request, response, or command packet. For request packets the two most significant bits of the 40-bit address are decoded to select one of four partition registers to access the 24 bit partition address - the 2 most significant bits in the 40-bit address used to address the partition register are reset in parallel with the access to the partition register. Now we need three pieces of information: 1) the extended 24-bit address to form an HT

read request packet with extended address, 2) the MAC address of the destination bridge to encapsulate the extended HT packet into Ethernet, and 3) the local MAC address, according to Ethernet frame format to enable the response. Item 3 has been set during initialization, and access to the source MAC address is not in the critical path. Items 1 and 2 have a direct correspondence among them (given a destination node ID or equivalently the remote partition address there is a unique MAC address associated with it). Therefore, the partition register can store both the 24-bit partition address (or larger as discussed above), and the destination MAC address together, thus reducing access time when forming the Ethernet frame. Once the remote MAC address and the 64-bit address have been found in the partition table, the new HT packet is constructed and encapsulated in a standard Ethernet packet. The encapsulated packet is then buffered until it can be sent using the local node's Ethernet MAC and the physical Ethernet interface. For packets that send a set amount of data, the control and data packets must be buffered until all the data has been encapsulated into Ethernet frames.

The receive behavior of the bridge on the remote node will require a "response matching" table where it will store, for every non-posted HT request (request that requires a response), all the information required to route the response back to the source when it arrives. Since the formats of HT request and response packets differ and this implementation desires not to change local HT operation, the SrcTag field of each packet is used to match MAC addresses from an incoming request packet with an outgoing response packet. Note that each request packet contains the source MAC address, and this is the address stored in the "response matching" table and later used as destination MAC address for the corresponding response. Encapsulation and buffering occur once again until the response and data can be transmitted over Ethernet.

It should also be noted that since HT SrcTags are 5 bits, a maximum of 32 outstanding requests can be handled concurrently by this approach. If two request packets arrive with the same SrcTag, then the latter packet is remapped before being stored in the table. When the corresponding response leaves the HT-Ethernet bridge, the SrcTag is mapped back to its original value to ensure proper HT routing on the requesting local node.

Once the response reaches the local HT-Ethernet bridge that initiated the read request, the HT packet is removed from its Ethernet encapsulation. The UnitID is changed again to that of the local host bridge and the bridge bit is set to send the packet upstream. This allows the local host bridge to route responses to the originating HT device.

Other transactions such as a posted write or a non-posted write involve similar sequences of events. The differences in these transactions are that for posted writes, no data is stored to create a response. For non-posted writes, only a "TargetDone" response is returned and no data needs to be buffered before the response is sent over Ethernet. Similarly, atomic Read Modify Write commands can be treated as non-posted write commands for the purposes of our model.

3.2.2 HToE Performance Results

Early synthesis tests using Xilinx software have indicated that the four modules that make up the bridge are individually capable of speeds in excess of 200 MHz – combined, unoptimized results indicate that the HT bridge is more than capable of feeding a 1 Gbps or faster Ethernet adapter with a maximum speed of 166 MHz. Additionally, estimates using a conservative 125 Mhz (1 Gbps) clock speed and evaluations for each of the request and reply critical paths suggest that the

latency overhead of such a bridge is on the order of 20-40 ns. In a Xilinx Virtex 4 FX60 FPGA, an unoptimized placement of the bridge uses approximately 1300-1500 slices, or approximately 5-6% of the chip. These preliminary statistics point to the HToE bridge being a small, fast implementation that is appropriate for investigating high-performance D-PGAS models.

Our current efforts are focused on stabilization and tuning of the implementation pending implementation on a hardware prototype.

4 Research Directions

The prototype HToE implementation provides a test bed to investigate a number of issues surrounding scalable PGAS systems. Several specific issues are described below.

4.1 D-PGAS Programming Interface

Our initial programming interface is inspired by the Berkeley GASNet [4] specification. Our long-term goals include working towards a native implementation of D-PGAS where the primitives should be implementable via extensions to the core instruction set architecture. The basic operations will be identified and implemented via new instructions or short sequences of existing instructions. The evaluation of the instruction set extensions will be via simulation while the functional evaluation will be in software, for example via the message handlers of GASNet. Several classes of operations that this interface should support beyond the obvious (read/write transactions) are immediately evident and are briefly touched on below – message ordering constraints, memory consistency, and synchronization.

Ordering constraints typically arise to ensure correctness (e.g., deadlock, coherence, etc.) of the protocol – in this case corresponding to the tightly coupled NI e.g., HT. These ordering constraints on messages must be honored when communication is extended across many nodes. In particular, HyperTransport specifies certain ordering constraints between subsets of transactions originating from a device, and in fact some of these constraints are specified or recorded in the packet headers. The HT protocol specification was designed for a specific target size and set of topologies (via HT bridges as described in the specification) and the ordering constraints rely on the anticipated implementation architectures which our proposal seeks to change. Preservation of these behaviors across a switched network is not straightforward. The situation is in fact analogous to the development of the Advanced Switching Interconnect for extending PCIe across multiple nodes.

A second class of constraints that must be honored is those arising from the memory consistency model. Ideally we would like to support several memory consistency behaviors using the primitives defined in the local protocol. Operations defined within HyperTransport for this purpose include fence and flush operations. The most efficient way to implement these types of messages and to conform to HyperTransport deadlock freedom requirements is to provide support in the bridge to handle and prioritize consistency-related packets. For example, a flush or fence operation normally pushes posted writes in a local HT system to memory so that all following reads will have the most up-to-date information. Each operation only operates on posted writes, but they allow for consistency on a per-transaction basis (flush) or on a per-device basis (fence). Extending these operations across a global address space requires additional support at the end points. One option is the following – use of remote-flush and remote-fence

operations. This would involve using a special posted write packet addressed to the local bridge, which then would generate a remote-flush or remote-fence command to be transmitted to the receiver. When the remote-flush or remote-fence command reaches the remote node, the remote bridge correctly interprets and propagates the correct local flush or fence command so that it operates like a locally generated HyperTransport command. In effect this model is one wherein the HyperTransport segment at the originating node is logically (and transparently) extended across the network.

The availability of remote atomic operations is necessary for coordinating multiple requesters. In the case of HyperTransport, the read-modify-write transaction is available to implement synchronized transfers. Further, we anticipate that remote-fence and remote-flush operations in conjunction with non-posted communication can be used to affect fine grained as well as coarse grained synchronized transfers - in fact to implement behaviors such as `wait` and `try` in the GASNet specification. This aspect of our work will proceed initially with the definition and adoption of a memory consistency model followed by the definition of the base primitives for the prototype implementation of D-PGAS.

4.2 Scalable Coherence

The prevalent shared memory coherence protocols have inherent barriers to scalability primarily because they possess two characteristics – i) uniform treatment of accesses to any memory address, and ii) application agnostic behavior in that all memory references are treated equivalently. Mechanisms such as snooping, flooding and the use of directories are global mechanisms for synchronizing updates to shared variables. By enabling non-uniform treatment of memory references or memory regions, one can develop approaches to coherence that are tailored to the reference behavior of the applications and the sharing pattern amongst cores (equivalently memory controllers). Such limited, dynamic, or on-demand coherence can be effectively supported via the flexible global address space mapping mechanisms described here supported by optimized page management policies. The research direction here begins with revisiting past research in scalable coherence protocols as a prelude to an effort focused on limited, dynamic, page-based, on-demand coherence protocols. For example, the address space mappings implicitly define the set of MCs that must participate in coherence at any given point in time and one can envision that MCs have two modes of operation – coherence mode and standard mode where the former requires checks that incur an additional cycle or more of latency. Furthermore, we plan to consider replacing the Ethernet prototype with a many-core network. We can consider support within the network for dynamically created coherence domains (partitions of the system) whereby coherence is maintained only within the current constructed coherence domain. Required functionality includes flexible data structures that can be tailored to the current set of pages that are to be coherent and managing traffic accordingly without penalizing non-coherent traffic.

4.3 Memory Latency Optimizations

Critical to remote access latency is the path between the wire and the DRAM. Tightly coupling the NI with the memory controller is a step towards reduction in remote latency. However, the DRAM memory access latency is a major component of this overall end-to-end latency and DRAM access latency is a function of the organization and capacity of DRAM. The interesting research question here is whether one can trade DRAM access latency for network latency, i.e., it

may be preferable to reduce the size of a DRAM partition at the expense of increasing the number of partitions and therefore nodes on the network. For example, if one can recover 75 ns by using smaller partitions but increase the number of network hops by 2-4 links at a cost of 2-4 ns/link, this may be a good tradeoff. We are also interested in the microarchitecture of the interconnect between the wire, memory controllers and local cache/core, with the goal of ensuring the fastest path from/to the memory controller and the wire, in this case HT.

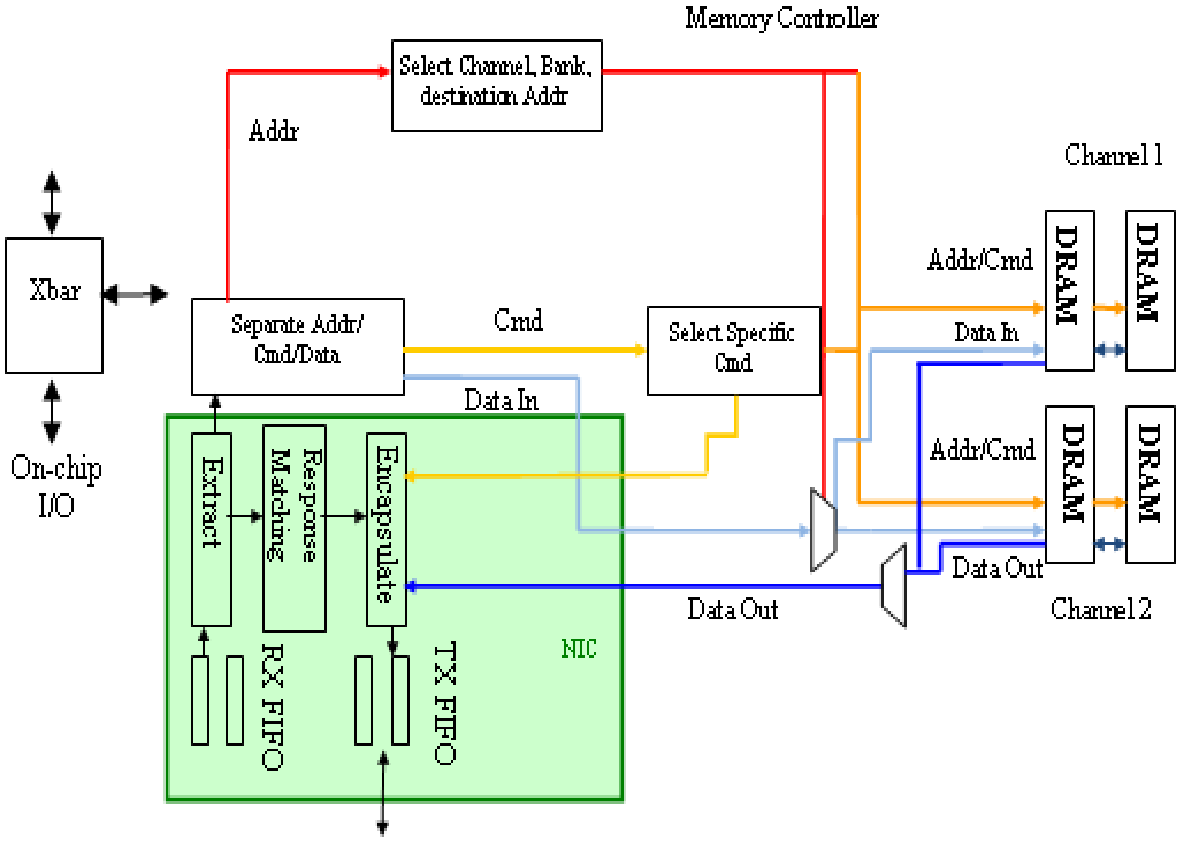

Figure 4: Integrated Memory Controller and NIC

Figure 4 illustrates an integrated DDR memory controller and queue-pair based NI. The example DDR controller excludes pre-fetch logic, ECC checking, etc. but captures the basic interface issues. The response matching table is necessary to keep information regarding remote requests such as the request source address, which is accessed during response generation. Other messages may not access the response matching table. For example, remote store operations with posted semantics will not generate responses and therefore will not need to store the source address of the request. Finally, this integrated MC-NI must provide support (not shown) for remote atomic operations which will be necessary for concurrently serving multiple remote requests and models such as fine grained RDMA communication models. Note that a local memory controller can share a partition with remote nodes without local host interaction.

Such integration is particularly useful when using FPGA co-processor accelerators where the on-FPGA memory controller is integrated with the host-interface to enable shared memory between the host and the FPGA. In our model, the FPGA DRAM will be mapped into the system wide

address space making the FPGA accessible to all nodes in the system and we hypothesize that this will enable efficient sharing of the FPGA as well as optimization of data transfers between the host and FPGA DRAM. Such aspects currently are being explored.

4.4 Message Passing

Of significant importance is efficient support for message passing. In the implementation proposed here header construction is performed in hardware and there is efficient support for zero-copy implementations. Of particular interest is the manner in which the semantics of standardized layers such as MPI and remote DMA (RDMA) can be supported. A useful place to start is with porting low-level software layers that support the PGAS model such as Berkeley's GASNet [4]. We propose the pursuit of an implementation of GASNet over our HToE prototype as a vehicle for benchmarking and measurement.

The availability of a D-PGAS system enables us to revisit communication optimizations such as pull messaging where the availability of data is posted or communicated (possibly via a message) and the data transfer is scheduled and affected by the receiver. Working with Ohio State University, we also plan to explore the implementation and optimization of MPI over D-PGAS.

4.5 Memory Provisioning for VMs

An application (or process) now is allocated a physical address space whose partitions may span multiple MCs, i.e., local and remote partitions, and therefore its virtual address space maps to MCs distributed across the system. This ability to aggregate physical memory resources across a machine via a low latency interconnect fabric is central to ability to consolidate and manage memory resources in the construction and deployment of virtual machines (VMs). The Virtual Machine Monitor (VMM) or hypervisor is responsible for allocating memory to VMs and as necessary updating the address translation tables in the integrated NI-MC to reflect the physical addresses accessible by the VM. Therefore VMs need no longer be limited by the memory at a single node, nor is the number of VMs supported at a node a function of the size of the local memory. We anticipate that this can be particularly valuable for data intensive applications and those that benefit from in-core databases since access to remote DRAM is several orders of magnitude faster than access to disk. Finally, the availability of global address spaces also presents some advantages for the migration VMs in a large scale system since the memory state will not have to be migrated.

4.6 Related Research

Distributed shared memory machines have been under investigation for the past two decades including the Stanford DASH and Flash multiprocessors [5], and SGI Origin [11]. These machines were driven by the need to support a large single *coherent* address space whereas at lowest level we are interested in first supporting a non-coherent address space over which smaller coherency domains can be dynamically defined and implemented as necessary. We are also motivated to find solutions compatible with commodity interconnects. Furthermore, the high productivity computing community is investigating PGAS solutions driven primarily from the perspective of productivity. PGAS languages, their optimized compilation, and efficient run-time execution are the object of several projects including UPC [2] and X10. We are particularly motivated to utilize the GASNet [4] primitives in support of our work. They include support for

fine-grained message-passing operations and have demonstrated the flexibility for data-intensive scientific computing operations that traditionally have driven the development and implementation of large cluster-based supercomputers. Utilizing this implementation is a natural point of departure for us.

5 Summary

This paper has advocated the use of a dynamic partitioned global address space (D-PGAS) for future high performance computing systems. Specifically, we argue that the advent of integrated high speed interconnects such as HyperTransport has created new opportunities for managing address spaces in these large cluster systems to improve performance and scalability and more than an effective and productive compilation and programming model. The model we have proposed differs from current models in that i) it is focused on physical addresses spaces, ii) is dynamically managed, and iii) seeks tight integration of memory controllers and networks. We are currently implementing an experimental version using Ethernet as the communication substrate to gather application specific information that will guide future architecture efforts.

6 References

1. B. Carlson, T. L. Ghazawi, R. Numrich, and K. Yelick, “Programming in the Partitioned Global Address Space Model,” Tutorial, Supercomputing 2003.

2. C. Bell, D. Bonachea, R. Nishtala, and K. Yelick. Optimizing Bandwidth Limited Problems Using One-Sided Communication and Overlap, IPDPS 2006,
   Tech report version: LBNL-59207.

3. Cray XT3 Data Sheet, www.cray.com

4. D. Bonachea. Gasnet Specification, version 1.1, Report No. UCB/CSD -02-1207, October 2002

5. D. H. J. e. a. Kuskin, J. Ofelt. The Stanford flash multiprocessor. ISCA ’98: 25 Years of the International Symposia on Computer Architecture (selected papers), pages 485–496, 1998.

6. D. Jessel, “AMD Operton Processors: A Better High End Solution,” AMD Boston Design Center, March 2005.

7. D. Slogsnat, A. Giese, and U. Brüning “A versatile, low latency HyperTransport core”, FPGA '07: Proceedings of the 2007 ACM/SIGDA 15th international symposium on Field programmable gate arrays, pg. 45-52, 2007

8. http://www.violin-memory.com/

9. Hypertransport 3.0 Specification, http://www.hypertransport.org/docs/tech/, April 26, 2007

10. InfiniBand Trade Association. InfiniBand Architecture Specification Volume 1. Release 1.2.1, Jan. 2008. http://www.infinibandta.org/specs/

11. J. Laudon, D. Lenoski, “The SGI Origin: A ccNUMA Highly Scalable Server” Proceedings of the 24th Annual International Symposium on Computer Architecture, 1997. pg. 241-251

12. J. Morrison and D. Turek, “The New Era of Supercomputing: Insights from the National Labs to Wall Street,” http://www-03.ibm.com/industries/financialservices/doc/content/bin/ibm_lanl_at_sifma_tech_2007_web.pdf

13. P. Charles, C. Grothoff, V. Saraswat, et. al, “X10: an object-oriented approach to non-uniform cluster computing”, OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN conference on Object oriented programming, systems, languages, and applications, pg. 519-538, 2005.

14. P. Conway, B. Hughes. The AMD Opteron Northbridge Architecture. IEEE Micro, 2007, v:27, n:2, pp:10-21