Towards a Scalable Field Programmable Gate Array Cluster for Interactive Parallel Ray-Tracing

Eoin Creedon, Ross Brennan and Michael Manzke
Interaction, Simulation and Graphics Lab, Department of Computer Science, Trinity College Dublin

Abstract
The Shared Memory and Message Passing programming paradigms have both been successfully investigated, initially for offline rendering and more recently for interactive parallel ray-tracing. Contemporary PC technology allows for the construction of clusters that can provide sufficient computational and communications resources for running interactive parallel ray-tracing applications. Furthermore, research has successfully demonstrated that Field Programmable Gate Arrays (FPGAs) may be used as a platform for these types of applications. However, large scale interactive ray-tracing tasks can exceed the available resources that can be provided by a single FPGA. This paper presents two scalable FPGA cluster architectures that could provide sufficient resources to execute even large scale interactive Ray-tracing applications. This is work in progress with the aim to ultimately evaluate interactive parallel Ray-tracing techniques on the two proposed cluster architectures.

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Hardware Architecture

1. Introduction
FPGAs have been investigated for many applications as an alternative to CPUs and as reconfigurable co-processors that can be controlled and communicated with through a host system’s I/O bus. One advantage of the latter approach is that the logic that implements the co-processor’s functionality may be changed at run time in order to adapt the FPGAs logic to the application that is being executed on the host system’s CPU(s). This makes the FPGA based co-processor beneficial for a number of applications that can be executed on a given host system. Callanan et al. [CGNP06] provide an example of a single FPGA co-processor solution that implements a lattice Quantum Chromodynamics (QCD) application on an FPGA using Handel-C and logarithmic arithmetic. Their results show that FPGAs and logarithmic arithmetic are a viable compute-platform for high performance computing.

The fact that FPGA logic can be adapted to the changing demands of a particular application introduces a flexibility that cannot be provided by Application Specific Integrated Circuits (ASICs). Even though an ASIC is likely to outperform an equivalent FPGA implementation, FPGA based applications have the potential to achieve high performance despite their relatively low clock frequencies by exploiting parallelism in the application. The FPGA may implement sections of an algorithm in concurrently operating digital logic blocks while read and write latencies to external memory may be hidden by overlapping memory communication with computation.

In their paper, Schmittler et al. [SWW04] present an interactive FPGA based ray-tracer that is suitable for dynamic scenes. This work is an FPGA specific implementation of earlier simulation work that targeted an ASIC implementation [SWS02]. Their FPGA based prototype implements a full ray tracing pipeline on a single FPGA and delivers 20 to 60 fps over a number of 3D scenes. The single FPGA based co-processor is a viable solution for many applications including ray-tracing, but once the FPGA resources have been consumed one must resort to using multiple FPGAs. Consequently a suitable interconnect must be chosen. A Printed Circuit Board could hold a limited number of FPGAs that are interconnected on the board but this solution exhibits a constant computational limit. A scalable solution, such as an architecture that could adapt its computational resources to the demands of the interactive ray-tracing application, is preferable. This objective could be achieved using a network
of FPGA boards. The number of FPGA boards would determine the computational resources that are available to the parallel ray-tracing application. The choice of interconnect would not only determine the available bandwidth and latencies but also the scalability of the system as well as its programming paradigm. Patel et al. [PMS*06] discuss various scalable FPGA-based multiprocessor architecture options.

In this paper we propose to evaluate two alternative FPGA cluster interconnects that target interactive parallel ray-tracing applications. The first choice is Ethernet. This interconnect is widely available on commodity FPGA boards that can hold sizable FPGAs and large amounts of external memory. The second interconnect should implement Distributed Shared Memory (DSM) in hardware that would also allow the interconnection of the DSM-FPGA cluster to host PCs. We have selected the Scalable Coherent Interconnect (SCI) [IEE92] for this purpose because it meets these design objectives and it exhibits high bandwidth and low latencies.

These two interconnects determine the respective programming paradigm and the ray-tracing algorithm. The hardware DSM-FPGA cluster should apply the shared memory programming paradigm for a parallel ray-tracing algorithm that is mapped onto the distributed FPGAs, bearing in mind that the design entry of the FPGA logic is performed through a hardware description language. On the other hand, the low cost Ethernet-FPGA cluster’s only option is to rely on the Message Passing paradigm if we disregard a software DSM system as a viable option for this architecture. Saldaña and Chow [SC06] provide some insight into FPGA based message passing but do not implement this in hardware.

The parallel ray-tracing algorithm is determined by the bandwidth and latencies of the FPGA cluster’s interconnect. This also influences the maximum complexity of the scene that can be processed. An interactive parallel ray-tracing algorithm may be implemented in two ways [Wal04]: Firstly through object space subdivision and secondly through screen space subdivision. Object space subdivision allows for the distribution of the scene database over all the external memories on the FPGA cluster boards. This approach requires the communication of rays if the next spatial partition is pierced by a ray and requires very high bandwidth. The hardware DSM-FPGA cluster is the most suitable target for this algorithm. The low cost Ethernet FPGA cluster should use screen space subdivision because it requires far less bandwidth but the entire scene database must be stored on every node in the FPGA cluster. This restricts the size of the scene database to the size of the external memory of a single node.

2. Hardware Description

We describe two approaches towards implementing a scalable FPGA cluster for interactive parallel ray-tracing. The first approach describes an implementation of a cluster using commercially available FPGA boards while the second approach describes a similar implementation using dedicated, custom built FPGA boards.

2.1. Background

In order to efficiently parallelize any application we need to be able to exploit the concurrency of multiple processing elements (PEs) running in hardware. Our objective of implementing a parallel ray-tracing algorithm in hardware can be achieved using FPGAs to enable the parallel execution of PEs fundamental to the ray-tracing application such as the computation of traversal, transformations and intersections. These ray-tracing PEs are defined using a Hardware Description Language such as VHDL or Verilog to leverage the fundamentally parallel nature of the underlying hardware. This technique however, is limited by the size and complexity of current commercially available FPGAs. Table 2 details the amount of single-precision floating point cores, which were used as example PEs, that can fit in a selection of current, commercially available FPGAs.

The most obvious way to increase the number of available ray-tracing PEs is to employ multiple FPGAs, which are physically connected together, using a bus based or crossbar interface on a single customized Printed Circuit Board (PCB). Each FPGA can run multiple PEs, which execute portions of the ray-tracing application in parallel. The PEs in multiple FPGAs can further communicate with one another, increasing the parallelism of the system. MIT’s BEE2 [BCW05] and the University of Southern California’s SLAACv2 [SCP’99] are two examples of projects, which already use this technique to great effect.

The problem with this methodology is the lack of scalability due to the fact that a PCB can only support a maximum number of components for various physical reasons, including increased wiring complexity, power supply demands and hardware costs. The easiest way to create a scalable system is by interconnecting multiple PCB nodes and to employ multiple PEs running across these nodes in order to create a fully scalable hardware architecture. This setup can be implemented using either commercially available FPGA boards or custom built PCBs.

We will evaluate two commercially available interconnect standards. Ethernet and SCI. The SCI interconnect implements a Distributed Shared Memory (DSM) in hardware while Ethernet does not. Consequently, an SCI implementation can be considered to be more tightly-coupled than an Ethernet implementation as it provides a shared-memory environment, higher bandwidth and lower latencies than Ethernet.

Ethernet provides a method of creating a cost-effective commodity based implementation of a loosely coupled cluster with distributed memory. This implementation would not be as scalable as a tightly coupled cluster implementation.
with distributed shared memory, but has the advantage of being able to utilize readily available commodity FPGA boards as nodes in the cluster. A hardware Distributed Shared Memory (HW-DSM) cluster, based on a commercially available interconnect technology such as SCI, would provide a more closely coupled and scalable implementation but would require the use of custom built PCBs and as a result would be more expensive to implement.

2.2. Overview

Several common architectural features exist between the two cluster approaches we describe, as indicated by figure 1. Both approaches require the PEs, which implement the distributed ray-tracing algorithm, to run in parallel in reconfigurable hardware. The major differences between our tightly and loosely coupled implementations occur in how the PEs access remote memory locations across the interconnect.

Figure 1(a) shows an example of how an Ethernet based system could be implemented. In this model, each node has its own local memory store and there is no concept of a global memory address space. Data may be communicated between the FPGAs by transferring data held in internal FPGA register implementations. For a send to complete correctly an appropriate receive must be present on at least one node. If an appropriate receive exists, the data is copied from the local buffer of the sending FPGA to a local buffer of the receiving FPGA before it can be accessed by the local requesting PE. The latency and increased contention limit an Ethernet interconnected system's scalability. The bandwidth restrictions of the Ethernet connection will not allow the propagation of all the rays between the nodes. Consequently the local node is required to hold the entire scene-model in its local external memory. The size of the external memory restricts the size of the scenes that can be operated on.

Figure 1(b) shows how a custom HW-DSM system could be achieved using an SCI interconnect. The SCI interconnect creates a shared memory environment in which every node connected to the interconnect can make hardware references to read and write memory locations on every other node connected to the interconnect. Each node in the system has its own local memory store which is made available to the shared memory address space. When a node requires data that is not available in its local memory, it makes a hardware memory address reference to global shared memory space in order to fetch the required data. The SCI Link Controller (LC) devices, which make up part of the SCI interconnect, are then responsible for routing the request to the correct node on the SCI interconnect. Once a request to the node with the required data has been made, that node will respond to the original requester node with the appropriate data. This low-level routing is hidden from the requesting application PE so, as far as the PE is concerned, it only has to make requests to read and write data. As this system implements a Non-Uniform Memory Access (NUMA) cluster, there will be larger latencies involved in reading and writing data to remote nodes however, by ensuring correct process synchronization and employing standard latency hiding methods such as pre-fetching and block-transfers, the delay in accessing remote nodes can be substantially reduced. As a consequence of this high-bandwidth, low-latency interconnect used to create a HW-DSM NUMA solution, the scene-model that is being ray-traced can easily be divided across multiple nodes. This implementation would be suitable to problems with larger scene-models as more nodes could be used to scale the cluster to the required problem size because

\[ \text{Figure 1: Comparative system overview between an Ethernet based and a Shared Memory based implementation.} \]
the efficiency of the interconnect does not degrade as more nodes are added.

2.3. Ethernet

Ethernet is the ubiquitous technology for connecting computers together for the purposes of creating Local Area Networks and is defined in the IEEE 802.3 standard [IEEE02]. Ethernet can be implemented as both a point-to-point and broadcast network technology. Point-to-point communication is enabled by switch technology, which allows connected nodes to send data packets to each other. Collisions may occur in this configuration though and as a result reduce the effective bandwidth between connected nodes. This limits the scalability of Ethernet compared with other interconnect technologies such as Myrinet and InfiniBand.

There are two main approaches for implementing an Ethernet network using an FPGA. They are an off-FPGA Ethernet Media Access Controller (MAC) chip or an on-FPGA Ethernet MAC. The Ethernet MAC connects to a commodity Ethernet physical layer chip, which is responsible for the transfer of the data across the actual physical wiring. In both cases an Ethernet control unit is required to interface with and operate the MAC. The ray-tracing application interacts with the control unit in turn giving it access to the network interconnect. This direct connection of the application with the network interconnect reduces the overheads that are normally present in an instruction set processor, such as device drivers and protocol stacks.

2.4. SCI

SCI is an ANSI/IEEE standard that defines a high performance interconnect technology, which is used to create a shared memory environment, providing solutions for a wide range of applications [IEE92]. It is a well established technology that defines a unidirectional point-to-point communication between neighboring nodes and is designed to scale well as the number of attached nodes increases, allowing for up to 64K nodes to be interconnected together at a peak transfer rate of 1 GB/s point-to-point.

Work on SCI was originally started as part of the Futurebus+ project in 1988. It effectively simulates a bus by providing bus-like services to the attached devices. The specification was finished in 1991 and became a formal IEEE/ANSI standard in 1992.

The SCI addressing scheme uses a 64-bit fixed addressing model, using 48-bits as a node offset address and 16-bits for the node address. Memory references made by one node, using either PIO or DMA, are translated into an SCI transaction by the SCI Link Controller Chips (LC) [Dol04] and are transported across the SCI fabric to the correct remote node. The remote node then translates this transaction into a local memory access, thus providing a hardware DSM implementation.

The SCI B-Link bus [Dol00] is used to interface the LC chips with the FPGAs. The B-Link protocol allows for up to 7 devices to be attached to a single B-Link bus however, in our HW-DSM implementation, two LC devices are connected to the FPGA using this bus, allowing for the construction of a 2D torus SCI mesh as each LC device interfaces with one individual ring. Packets are routed over the B-Link to the correct LC device and then forwarded across the SCI fabric to the appropriate node according to a routing table, which is configured during SCI fabric initialisation. This enables distributed scalable routing of SCI packets between in-

---

(c) The Eurographics Association 2006.
2.5. Topologies

Ethernet does not specify one standard topology that it can be used as, though the normal configuration is for all machines to be connected to each other through a switch in a star topology. In our loosely-coupled cluster implementation the star topology will be used to allow every node a direct connection to every other node in the cluster as outlined in figure 2(a).

SCI defines an interface standard that enables the use of many different interconnect configurations ranging from simple rings and tori to complete multistage switched networks. Three common non-switched topologies are a ring (1D torus), 2D torus and 3D torus. In our HW-DSM implementation, we will employ a 2D-torus topology. Figure 2(b) outlines a setup where one ring of the torus is used to connect host systems to the cluster, while the second ring is used to connect the custom built nodes together.

3. Communications

Two main programming models exist for communication across clusters. These are the Message Passing and Shared Memory Models.

3.1. Message Passing Model

Of the two network paradigms being looked at, Message Passing is better suited to an Ethernet configuration. Message Passing primitives are provided for the sending, receiving and synchronization. These primitives should be easily integratable with the ray-tracing hardware while also being light-weight and efficient.

To maximize computation the message passing paradigm allows the interleaving of computation and communication through explicit message calls. On the FPGA based system this functionality can be performed in parallel as the computation logic and the communication logic are separate hardware entities.

3.2. Message Passing API

The basic functionality that is required with a message passing system is the ability to send data from one node and to receive it correctly and appropriately on another node [CSGP99]. Data dependencies between nodes and the possibility of nodes operating at different speeds or in different orders require that a synchronization mechanism is also provided. This synchronization can be either explicitly called (non-blocking) or is implicitly called (blocking) as part of the send and receive mechanism [SOHL’98]. Table 1 provides a list of the primitives that the API will provide.

3.2.1. Send message

Figure 3(a) shows the flow of operations for sending a message. To send a message, the programmer specifies the identity of the node they want to communicate with along with the size of the data that will be sent. Using this information, the sending mechanism builds the necessary send data structure. The send mechanism then commences the send by first telling the node it is sending data to that it has data to communicated. Once this is acknowledged by the receiving node, the message is transmitted and barring errors the send completes successfully allowing the sending node to reuse the sent memory locations. The ray-tracing application is then informed of the successful completion of the sending.

3.2.2. Receive message

Figure 3(b) shows the flow for receiving a message. A message is only receivable if the programmer has requested a receive. Two types of receives are possible, a wild-card receive resulting in receiving data from any node and a point-to-point receive meaning that data from only a particular source is accepted. To receive data, the programmer specifies which node the data is to come from and other order identification tag data to ensure the messages are received in order and as expected. Once the receive mechanism is aware a send is present, it acknowledges this and begins receiving the data from the sending node. If an error occurs during the transmission, the receiving node informs the sending node which must then re-send the message. On a successful receive, the ray-tracing application is informed which then takes the appropriate actions.

3.2.3. Synchronization

Two versions of synchronization are possible, one implicit, the other explicit. Both implementations though are still

© The Eurographics Association 2006.
Host/Hardware

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP_Begin</td>
<td>Primitive to distribute the necessary start conditions</td>
</tr>
<tr>
<td>MP_End</td>
<td>Causes all nodes to enter a known finishing state</td>
</tr>
<tr>
<td>MP_Issue</td>
<td>Initial data distribution to the nodes</td>
</tr>
</tbody>
</table>

Hardware Only

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP_Send</td>
<td>Non-Blocking message send</td>
</tr>
<tr>
<td>MP_Send</td>
<td>Blocking message send</td>
</tr>
<tr>
<td>MP_Recv</td>
<td>Non-Blocking message receive</td>
</tr>
<tr>
<td>MP_BRecv</td>
<td>Blocking message receive</td>
</tr>
<tr>
<td>MP_Sync</td>
<td>Barrier call across multiple nodes</td>
</tr>
</tbody>
</table>

Table 1: Message Passing primitives

called by the programmer. The implicit version is called as part of the sending or receiving and is known as a blocking communication. This causes the application’s execution to be halted until the communication has been completed. This is useful to ensure correct flow control of messages over the nodes. Explicit synchronization allows for the communication and computation to proceed in parallel with each other. This results in a better use of the available system as the communication is usually the slower aspect. To prevent data corruption, the programmer must explicitly test and wait on the message to ensure that the communication has completed as expected.

3.2.4. Memory Interface

A separate memory interface entity is utilised to provide simple memory reads and writes. The message send and receive control units are associated with appropriate data buffers, as defined by the ray-tracing application. This configuration enables the sharing of buffers between the application and the communication structure resulting in a more efficient communication structure.

3.3. Shared Memory Model

In shared-memory models, all processing elements share a single address space and inter-node communication is achieved through shared-memory locations or messages deposited in shared-memory buffers. There is no requirement for the programmer to manage the explicit movement of data between nodes as the communications methods are indirect. Any processing element can theoretically make direct hardware memory references into the global address space without requiring knowledge about whether the memory location that it is reading from or writing to is situated locally or remotely. In reality though, it is still important for the processing element to know if it is accessing local or remote memory due to the increased latencies involved when accessing remote memory. Consequently, it is important to outline a set of basic shared-memory communications primitives which implement certain standard message passing features such as process locks and barriers. A method of hiding latencies from the processes must also be developed. This may include commonly used techniques such as transfer scheduling, block transfers and pre-fetching.

3.4. The Shared Memory API

For the purposes of our HW-DSM cluster implementation, the question of a shared memory API must be approached from two different angles. The first is from the perspective of the host machines, which are attempting to communicate with each other across the SCI fabric and also with the PEs running in the FPGAs of the custom built nodes. The second is from the perspective of the PEs, running in hardware inside the FPGAs, attempting to communicate with one another across the SCI fabric as well as with the host nodes.

Software Infrastructure for SCI (SISCI) [SIS01, GAB+01] is the application programming interface that covers different aspects of how SCI interconnects can be accessed from host systems. It specifies the general functions, operations and data types made available as part of the SCI standard and takes care of mapping local address segments into the shared memory address space and for checking whether errors have occurred during data transfers. Low level communication among nodes is accommodated by a set of SCI transactions and protocols that include support for:

- Data read/write
- Cache coherence
- Synchronization and message passing primitives

While SISCI is useful for accessing SCI interconnects from host PCs, it needs to be adopted to a form suitable for use in embedded hardware systems such as the HW-DSM cluster that we propose. This part of the project has yet to be implemented but any hardware based API that needs to be created would have to provide hooks for integration with the standard SISCI API, while at the same time adhering to standard threaded multiprocessing techniques for data transfer and process synchronization. This custom API will be implemented in a HDL and will run in hardware allowing direct access to and control of the various PEs running in the FPGAs. Some examples of primitives that may be needed are given in table 3.

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SM_getNodeNum</td>
<td>Find the rank of the current PE</td>
</tr>
<tr>
<td>SM_getNodeCount</td>
<td>Returns the total number of PEs</td>
</tr>
<tr>
<td>SM_Lock</td>
<td>Create a Shared-Memory lock</td>
</tr>
<tr>
<td>SM_Unlock</td>
<td>Release a Shared-Memory lock</td>
</tr>
<tr>
<td>SM_Sync</td>
<td>Shared-Memory sync barrier</td>
</tr>
</tbody>
</table>

Table 3: Shared Memory primitives

© The Eurographics Association 2006.
4. Implementation

Table 2 provides details of some commercially available FPGA boards that can accommodate the Ethernet configuration. Some of the boards offer 10/100Mb Ethernet while others provide tri-mode Ethernet (10/100/1000Mb). Our custom board is present for comparative reasons and provides the SCI interconnect. The floating-point cores count is based on the amount of multiply-addition modules, along with control logic, that can be synthesized for the chip. This is given for comparative reasons to show the possible computation capabilities of the different FPGAs. Xilinx Single precision floating-point cores are used and Xilinx ISE 8.1 was used for the synthesis. The FLOPS calculations are based on the peak bandwidth.

From evaluating the data in table 2 it was decided to use the Xilinx XUP V2P board as this provides both good performance and high memory bandwidth. For this board the memory interface controller, an Ethernet MAC [Ope01] and control logic along with the floating point multiply-accumulate unit have been tested.

5. Conclusions and Future Work

The arguments that are presented here show the potential for implementing interactive ray-tracing on two different platforms when the required communication models are taken to each other and to the host nodes in the cluster through a commercially available Ethernet switch. Each node contains a DDR-SDRAM module which is connected to a DDR memory controller resident on the FPGA logic.

Figure 5 illustrates a prototype custom HW-DSM node, which has been designed at Trinity College Dublin [MBO06]. The prototype implements a Shared Memory system as described by figure 1(b). The node has two SCI LC devices, which allow the implementation of a 2D-torus topology, as well as local DDR-SDRAM. In this particular implementation, the bridge functionality is performed by a dedicated FPGA (linking local memory and the shared memory interconnect), while a second FPGA is used to run the desired application processing elements that implement the ray-tracing algorithm. The second (application) FPGA has a dedicated link to the (bridge) FPGA and in this way can interface with local and shared memory address space.
into account. From looking at the possible implementations the size of the model has a large influence on which system platform is appropriate. From here, the direction of the work will involve finalizing the communication primitives and implementing the ray-tracing application on the FPGA within the two models that have been proposed.

6. Acknowledgments

The HW-DSM aspect of this work was funded by Science Foundation Ireland Basic Research Grant 04/BRG/CS0350. We would also like to thank Dolphin Interconnect Solutions, Xilinx and ATI for their support in this work.

References


[Dol00] Dolpin Interconnect Solutions: A Backside Link (B-Link) for Scalable Coherent Interface (SCI) nodes., September 2000.


