A SCALABLE AND RECONFIGURABLE SHARED-MEMORY GRAPHICS CLUSTER ARCHITECTURE

Ross Brennan, Michael Manzke, Keith O'Conor, John Dingliana and Carol O'Sullivan

Graphics, Vision and Visualisation Group,
Department of Computer Science, Trinity College Dublin

Abstract: ***If the computational demands of an interactive graphics rendering application cannot be met by a single commodity Graphics Processing Unit (GPU), multiple graphics accelerators may be utilised on multi-GPU based systems such as SLI [1] or Crossfire [2] or by a cluster of PCs in conjunction with a software infrastructure. Typically these PC cluster solutions allow the application programmer to use a standard OpenGL API. In this paper we describe an FPGA based hardware architecture, which provides an interface for multiple commodity graphics accelerators. Our scalable parallel rendering architecture aims to accelerate graphics applications using a tightly coupled hybrid system of parallel commodity GPUs and reconfigurable hardware [3], while providing similar services to the above mentioned approach. This is work in progress. So far, we have designed and manufactured the required custom-hardware. Currently, we are focussing on implementing the shared-memory subsystem.***

Keywords: FPGA, Shared-memory, Hardware-DSM, SCI, GPU, PCB

1. INTRODUCTION

In this paper, we describe a scalable tightly coupled cluster of custom-built boards that can provide an interface for commodity graphics accelerators. These boards are supplied with rendering instructions by a cluster of commodity PCs that execute OpenGL graphics applications. The commodity PCs and custom-built boards are interconnected with an implementation of the IEEE 1596-1992 Scalable Coherent Interface (SCI) [4] standard. This technology provides the system with a high bandwidth, low latency, point-to-point interconnect. Our design allows for the implementation of a 2D torus topology with good scalability properties and excellent suitability for parallel rendering. Most importantly, the interconnect implements a Distributed Shared Memory (DSM) architecture in hardware. Figure 1 shows how local memories on the custom-built boards and the commodity PCs become part of the system wide DSM using the SCI interconnect. The FPGAs assist the SCI implementation while providing substantial additional computational resources that may be used to control the GPUs. These reconfigurable components are an integral part of the scalable shared-memory graphics cluster and consequently increase the programmability of the parallel rendering system in the same way that vertex and pixel shaders have increased the programmability of graphics pipelines.

Funded by the SFI: Basic Research Grant 04/BRG/CS0350

Current scalable high-performance graphics systems are either constructed using special purpose graphics acceleration hardware or built as a cluster of commodity components with a software infrastructure that can exploit multiple graphics cards [5, 6]. These solutions are used in application domains where computational demand cannot be met by a single commodity graphics card e.g., large-scale scientific visualisation. The former approach tends to provide the highest performance but is expensive because it requires frequent redesign of the special purpose graphics acceleration hardware in order to maintain a performance advantage over commodity graphics hardware. The latter approach, while more affordable and scalable, has intrinsic performance drawbacks due to the computationally expensive communication between the individual graphics pipelines. The advent of recent technologies by ATI and nVidia also allows for the creation multi-GPU systems. Our hybrid architecture aims to bridge the gap between all of these solutions by offering a minimal custom-built hardware component together with a novel and efficient shared memory infrastructure that can exploit modern consumer graphics hardware.

This paper is structured as follows: In section 2 we discuss related work. This is followed by a description of our cluster design in section 3. Section 4 details the interconnect technology that will transform the individual cluster nodes into a hardware DSM. Section 5 provides information about the custom-built GPU-FPGA cluster nodes, including a description of our custom-hardware design. Section 6 describes the proposed design of the software for the commodity cluster and the logic for the FPGAs that will turn our proposed architecture into an efficient parallel rendering system. A final discussion of our design is provided in section 7.

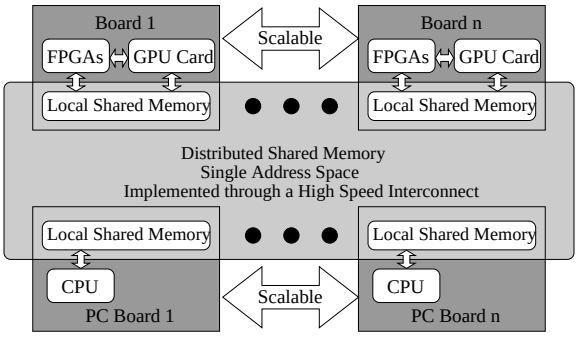

Fig. 1. Shared Memory.

2. RELATED WORK

This work is motivated by research conducted as part of the WireGL [5] and Chromium [6] projects. Our parallel rendering solution integrates FPGAs into the graphics pipeline as a distinct computational stage between the application stage, that is executed on the host system's instruction set processor, and the geometry stage of the commodity graphics accelerator. Recent work at Saarland University demonstrated an implementation of a real-time ray tracer on a single FPGA that would otherwise require a cluster of commodity PCs [7]. Other recent work at the University of North Carolina implemented an FPGA based view-independent graphics rendering architecture [8] and earlier research at the University of Tübinger developed a real-time volume renderer, which was implemented using DSP and FPGA components [9]. These are just some examples that demonstrate the suitability and computational ability of reconfigurable hardware for graphics applications. Our architecture further improves the performance potential of these components by making them part of the hardware DSM. A recent commercial shared memory graphics cluster solution, the Onyx4, was evaluated by the University of Utah [10], while another project there was concerned with the implementation of interactive ray tracing on clusters that applied a software DSM for the ray tracer [11, 12, 13].

nVidia's SLI [1] and AMD's Crossfire [2] systems provide hardware support for multi-GPU configurations but require specialised motherboard support and are currently not scalable beyond 4 GPUs per system.

Moerschell et al. have demonstrated how a directory-based coherence protocol could enforce coherence of texture memories in multiple GPUs that interface with shared-memory [14] in order to ease programming in multi-GPU environments. The implementation would be suitable for our scalable architecture.

3. CLUSTER ARCHITECTURE

Our scalable and reconfigurable shared-memory graphics cluster is predominantly constructed from commodity, off-the-shelf, components with a limited amount of custom-built hardware. The custom-built boards allow graphics accelerator cards to interface distributed shared-memory that is also shared by a number of PCs in the graphics cluster. Figures 1 and 2 show the overall design of our architecture. One of the main design objectives was to keep the custom-built hardware part of the system as small and simple as possible while still enabling high-performance computations. Adaptability to the latest generation of desktop graphics acceleration hardware was a further important issue, as the highly competitive nature of this market ensures that the performance of these cards increases dramatically with each new version. The ability to change parts of the hardware design, after the PCBs for the GPU-FPGA nodes have been manufactured and populated with Integrated Circuits (ICs), was significant in order to conduct research into different implementation alternatives. Reconfigurable hardware is an ideal solution to meet these design objectives, with the added advantage of providing substantial additional computational resources for the application stages of the parallel graphics pipelines. These additional resources can be used to implement algorithms, usually executed on the CPU or GPU of a desktop machine. These algorithms may be defined at compile time by the application developer and loaded into the reconfigurable hardware just as a traditional graphics application is loaded into the main memory.

3.1. Parallel Rendering and Sorting

The scalable shared-memory hybrid system of PCs, GPUs and Reconfigurable Hardware is a parallel rendering architecture and as such can be classified according to Molnar et al's taxonomy [15]. This classification defines parallel rendering as a sorting problem and divides the graphics pipelines into two main pipeline stages: geometry and rasterisation. The geometry processing stage is concerned with transformation, clipping and lighting and is parallelised by distributing subsets of the primitives in the scene over the available concurrent geometry stages. The rasterisation deals with scan-conversion, shading and visibility determination and the graphics application may take advantage of the parallel rasterisation stages by assigning each stage a share of the pixel calculations. Rasterisation is then followed by the image composition. In order to increase the utilisation of the different parallel pipeline stages, a sorting or redistribution of data between the main stages may be performed. Molnar et al's taxonomy specifies redistribution during the geometry processing as "sort-first", sorting between the geometry processing and rasterisation as "sort-middle" and a distribution during rasterisation as "sort-last".

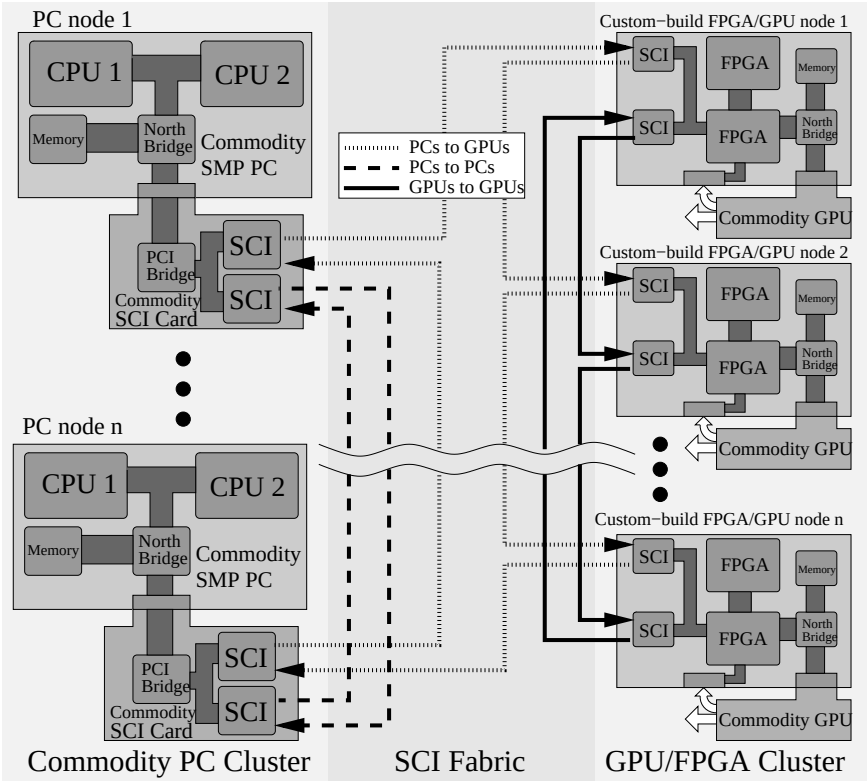

Fig. 2. Hybrid Parallel Graphic Cluster.

Some compute clusters such as Stanford's Chromium [6] utilise a software infrastructure to convert a cluster of commodity, off-the-shelf desktop machines with graphics acceleration cards into a parallel rendering system. Such systems can implement sort-first and sort-last alternatives but sort-last mechanisms can only be achieved through expensive read-backs of colour and depth buffers, since this is the only form of access to data in the graphics pipeline of a commodity GPU card. The sorting of data before it enters the different stages of the parallel graphics pipelines allows load balancing and therefore increases the utilisation of the overall system. An efficient solution for load balancing a sort-first algorithm is provided through pre-transformation in order to determine the most suitable data distribution over the GPUs [5]. These pre-transformations are part of the geometry processing stage and calculate the screen space position of primitives in order to allocate them to screen regions that are served by a particular GPU. This computation is one of the overheads that must be carried by a scalable parallel rendering system in order to most efficiently exploit parallelism.

3.2. Sorting Acceleration

Clusters that provide parallel rendering facilities through a software infrastructure must perform these pre-transformations with the assistance of the system's CPUs as part of the graphics pipeline's application stages. In contrast, our architecture provides the application stages with reconfigurable hardware resources on every GPU-FPGA node. The FPGAs interface the distributed shared-memory in the same way as the GPUs and CPUs are connected to the shared address space. This allows individual FPGAs to communicate with each other at extremely low latencies and high bandwidth ($\geq$500Mbytes/s). This additional infrastructure has the potential to implement complex sort-first load balancing mechanisms without any additional computational overhead for the CPUs and at superior performance. Furthermore, a sort-last implementation on the tightly coupled cluster of GPU-FPGA nodes will also be feasible. It is intended that these operations will be implemented with limited or no CPU involvement.

4. INTERCONNECT TECHNOLOGY

The interconnect, in conjunction with the reconfigurable hardware, fulfils a vital function in our design. The distributed FPGAs allow the pipeline's application stages to migrate subsets of their computations from the CPUs onto the FPGAs, while the interconnect implements the DSM in hardware with the aid of additional logic in the FPGAs. We used

the IEEE 1596-1992 Scalable Coherent Interface (SCI) [4] as the interconnect standard for this architecture. This system area interconnect standard defines a high bandwidth and low latency interconnect and is scalable to a large number of nodes while providing bus-like services. The interconnect allows for a flexible fabric configuration, which guarantees the scalability of the parallel rendering architecture.

4.1. SCI in Commodity Systems

Defined in 1992, SCI is a well established technology with many high performance cluster implementations employing it as an interconnect (e.g, PC2 University of Paderborn Germany, University Of Delaware, The Bartol Research Institute USA and National Supercomputer Centre in Sweden) [16]. Subsets of the SCI standards have been implemented and are available as commodity components. In particular, Dolphin [17] have implemented PCI cards that bridge PCI bus transactions to SCI transactions. Compute nodes with PCI slots may be interconnected through PCI-SCI bridges together with a suitable SCI fabric topology, thus bridging their PCI buses. Memory references made by one of these nodes into its own PCI address space are translated into a SCI transaction and transported to the correct remote node. The remote node translates this transaction into a memory access, thus providing a hardware DSM implementation. Programmed IO (PIO) and Direct Memory Access (DMA) may be performed without the need for system calls.

Figure 3 illustrates the design of a Symmetric Multiprocessor (SMP) node with a commodity SCI card in one of its PCI slots and also shows the main components on this SCI card. The PCI-SCI bridge translates between PCI transactions and SCI transactions and forwards them onto the PCI bus or the B-Link bus as appropriate. The SCI B-Link bus connects the PCI-SCI bridge with up to seven SCI Link Controllers (LCs) or alternative components. The SCI cards shown have two SCI Link Controllers attached to the B-Link and consequently are suitable for the construction of a 2D torus. Systems with more than one LC route packets over the B-Link to the correct LC according to an internal routing table. This enables distributed routing of SCI packets between individual SCI rings without an expensive central SCI switch. Routing is configured during SCI fabric initialisation. Every LC has an input and output port and the output port of one LC component is connected via a cable to the input port of another LC component. These 16-bit wide links are unidirectional with a bandwidth of 667Mbytes/s.

4.2. SCI Commodity System Software

SCI driver software and a SISCI API for the Dolphin SCI cards force the operating system to reserve a particular section of the main memory for SCI, thus also prohibiting paging of this set of memory pages [17]. The software then

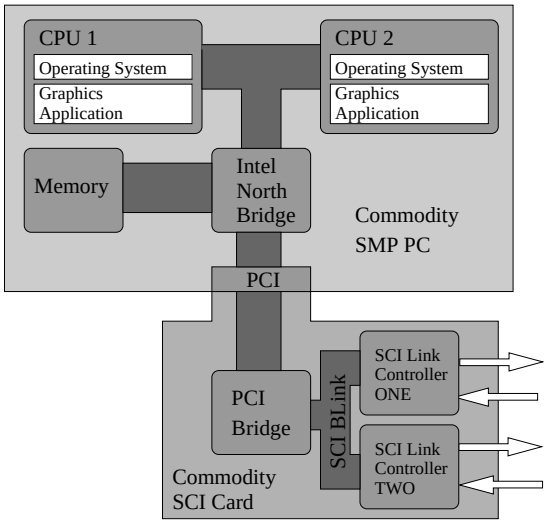

Fig. 3. SMP Desktop Node with 2D SCI-PCI interface card.

makes this part of the memory available to other nodes in the SCI cluster. The processes of remote nodes may then map this memory into their own virtual address space. Subsequent references to virtual addresses that are located on a remote node will be executed in hardware without the need for expensive system calls.

4.3. SCI and Real-Time Constraints

Concurrent rendering operations require synchronisation that must meet real time constraints. Furthermore, interactive immersive scientific visualisation frequently involves complex user input such as the tracking of the users motions. This motion-tracking data must be processed rapidly to influence the animation in real time, as large latencies between motion detection and animation can cause simulator sickness [18]. We postulate that a parallel rendering cluster interconnect with low latencies and deterministic behaviour may assist a cluster architecture in meeting these real time constraints. SCI technology is already being used in mission critical real-time applications. For example, Thales Airborne System employs SCI for backplane communication in their EMTI unit (Data Processing Modular Equipment). This scalable unit is integrated into the Mirage F1, 2000 and Rafale combat aircraft, NH-90 helicopters, Leclerc tanks, sub-marines, the Charles de Gaulle aircraft carrier and strategic missiles [19]. Manzke et al. have previously conducted research on SCI fabric's suitability for real-time application [20, 21].

5. COMMODITY AND CUSTOM-BUILT GPU-FPGA CLUSTER NODES

Figure 2 shows on the left side how a PC cluster is interconnected with commodity SCI cards. The SCI fabric then

connects the Commodity PC cluster to the custom-built GPU-FPGA cluster nodes on right side.

5.1. Cluster's Commodity Destop Machines

This part of the scalable parallel rendering cluster is completely assembled from commodity components and provides the services of a non-cache coherent hardware DSM. Increasing the number of nodes in combination with the possibility of changing the SCI fabric topology allows scalability to be achieved. These nodes execute the parts of the graphics pipeline's application stages that are not migrated onto the FPGAs on the GPU-FPGA nodes shown in Figure 4.

5.2. Cluster's custom-built GPU-FPGA Boards

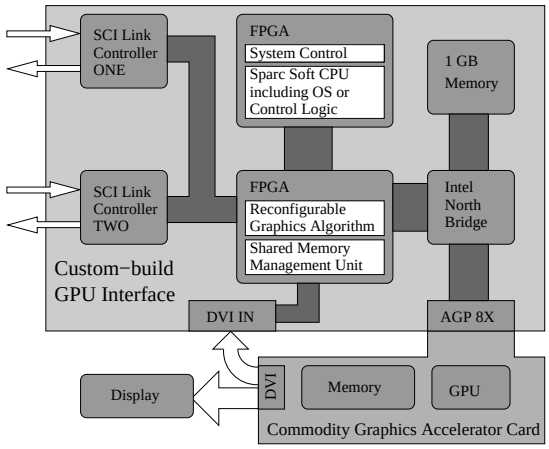

Fig. 4. GPU Cluster Node with a commodity graphics card in the AGP slot

The GPU-FPGA cluster nodes are custom-built boards that provide an AGPx8 interface for commodity graphics accelerator cards (See Figure 4). Two SCI Link Controllers (LCs) interface with a Xilinx XC2V2000 FPGA (Bridge-FPGA) via the 64-bit B-Link bus at 800 Mbytes/s. These LCs can be purchased from the manufacturer of the SCI cards [22] as Application Specific Integrated Circuits (ASICs) and solve many of the SCI link level implementation challenges. The Bridge-FPGA is connected to a second Xilinx XC2V1000 FPGA (Control-FPGA) via an Advanced Microcontroller Bus Architecture (AMBA) bus at 400 Mbytes/s [23] and an Intel i865GM chip (Northbridge) via a front-side bus (FSB) interface, which operates at 3.2 Gbytes/s. The Northbridge then provide a 2.7 Gbytes/s interface to 1 Gbyte of DDR memory and a 2.1 Gbytes/s link to the AGP slot. The DDR memory forms part of the global address space. It is the Bridge-FPGAs task to translate global address space memory references into SCI transactions and vice versa. Figure 5 shows an image of one of our first generation prototypes. This PCB has a 10 layer stack-up and 5908 pins and vias in order to mount 1277 components. Four prototypes of these boards have been manufactured at a cost of less than EUR2000 per board. It can be expected that this price will be significantly lower if manufactured in larger quantities.

Some of the services provided by the commodity SCI card must also be implemented in the Bridge-FPGA in order to allow the PCs to function with the GPU-FPGA cluster nodes. Management of the shared-memory address space is one of the main functions of the Bridge-FPGA. The second and equally important objective is to execute application-specific graphics algorithms as outlined previously. In our design, the memory and the graphics subsystems are directly connected to the SCI interface through the Northbridge and Bridge-FPGA. This approach avoids the bandwidth restrictions and additional latencies that are introduced in commodity SCI cards by the I/O bus.

The direct SCI connection of the GPU-FPGA nodes classifies these nodes as tightly-coupled where on the other hand the commodity SCI cards are less tightly-coupled. Nevertheless, the commodity SCI cards achieve approximately 300 Mbytes/s with a 1.5 μs application-to-application latency. The maximum bandwidth of the commodity SCI cards is dependent on the motherboard's chipset [17]. Measurements on two-node and four-node fabrics have demonstrated that FPGAs directly connected to the B-Link may exchange data at more than 500 Mbytes/s over the SCI fabric [24]. Latency measurements for this set-up are not available but it is reasonable to assume that transactions that do not have to pass the PCI bus will exhibit a significant low latency (< 1.5 μs).

6. FUTURE WORK

Beyond the design and implementation of the hardware architecture, a software infrastructure must be designed that best exploits the maximum performance potential of the scalable tightly-coupled cluster. We intend to implement parallel rendering services, while also taking advantage of the uniquely available reconfigurable hardware, to accelerate data distribution.

Like Chromium, our own system driver will intercept OpenGL calls, packing and redistributing them to the appropriate cluster nodes. However, we also hope to improve upon Chromium's infrastructure as the low-latency, high-bandwidth SCI interconnect will allow rapid distribution of large amounts of geometry and texture data while the Distributed Shared-Memory will keep data replication to a minimum. This should allow for any OpenGL application to run unaware of the cluster, however in order to leverage the full potential of the cluster an application should take advantage of all available reconfigurable hardware for distributing computational costs.

As the newer graphics cards are themselves programma-

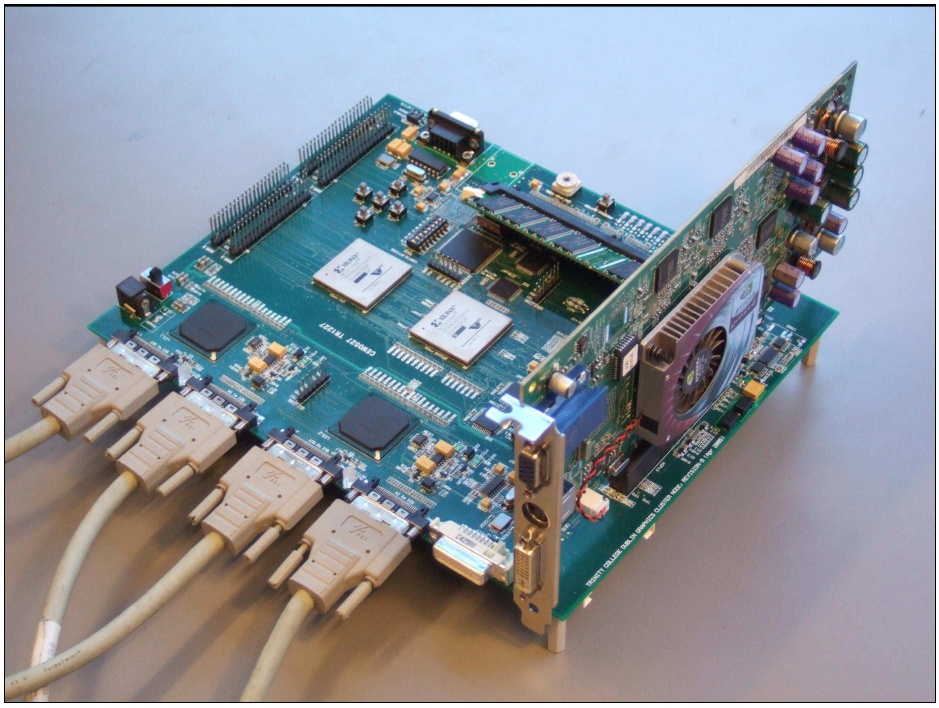

Fig. 5. shows the first prototype of our custom GPU-FPGA boards.

ble, this extra functionality will also be employed for accelerating suitable graphics algorithms via vertex and pixel shaders, where appropriate. Depending on the target application, the programmability of the different pipeline stages must be exploited with a specific approach. Interactive real time immersive scientific visualisation may require the reconfigurable hardware to implement efficient sort-first and sort-last mechanisms. On the other hand, video processing depends heavily on pixel shaders and may require the reconfigurable hardware to schedule images over the concurrent shaders and to perform some Digital Signal Processing (DSP).

In order to access the graphics adapters themselves without an OS-level driver to interpret API calls, a driver must also be implemented directly in reconfigurable hardware. This will be achieved by adapting the kernel-space functionality of open source drivers such as those of the Direct Rendering Infrastructure (DRI). This custom-built driver will be necessary in order to optimise GPU access to the clusters resources, in particular Distributed Shared Memory access for texture and geometry data.

The architecture of our design also makes it a suitable platform for other applications, such as large scale ray-tracing applications [25], due to the high-bandwidth, low-latency communications interconnect coupled with the availability of large amounts of local memory on each node that can be made available as part of the hardware-DSM.

7. CONCLUSIONS

In this paper, we have described how we designed a tightly coupled scalable Non-Uniform Memory Access (NUMA) architecture of distributed FPGAs, GPUs and memory using a limited amount of custom-built hardware. The first prototype boards are currently being tested and we expect that they will communicate data at 500Mbytes/s with low latencies ($< 1.5\ \mu$s). This parallel rendering cluster will connect to a commodity PC cluster that will be used execute the graphics application using the same high-speed interconnect. We have presented a comprehensive design for this novel architecture and estimate, based on the arguments presented, that this solution could outperform pure commodity implementations without increased hardware cost while maintaining its adaptability to the most recent generation of commodity graphic accelerators and target applications.

8. ACKNOWLEDGEMENTS

We would like to thank Dolphin and Xilinx for their support for the project through hardware and software donations. Dolphin also gave us valuable advice concerning our PCB design. This work was funded by the SFI Basic Research Grant 04/BRG/CS0350.

9. REFERENCES

[1] nVidia. http://www.slizone.com/page/home.html, 2007.

[2] AMD. http://ati.amd.com/technology/crossfire/, 2007.

[3] Michael Manzke, Ross Brennan, Keith O'Conor, John Dingliana, and Carol O'Sullivan. A Scalable and Reconfigurable Shared-Memory Graphics Architecture. In *SIGGRAPH '06: Material presented at the ACM SIGGRAPH 2006 conference*, page 182, 2006.

[4] The Institude of Electrical and Electronics Engineers, Inc. *IEEE Standard for Scalable Coherent Interface (SCI) 1596-1992*, 345 east 47th street, new york, ny 10017-2394, usa edition, 1993. ISBN 1-55937-222-2.

[5] Greg Humphreys, Matthew Eldridge, Ian Buck, Gordon Stoll, Matthew Everett, and Pat Hanrahan. WireGL: A Scalable Graphics System for Clusters. In *SIGGRAPH*, pages 129–140, 2001.

[6] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter D. Kirchner, and James T. Klosowski. Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters. In *SIGGRAPH*, pages 693–702, 2002.

[7] Jörgen Schmittler, Sven Woop, Daniel Wagner, Wolfgang Paul, and Philipp Slusallek. Realtime Ray-Tracing of Dynamic Scence on an FPGA Chip. In M. McCool T. Akenine-Möller, editor, *Graphics Hardware 2004*, 2004.

[8] J. Stewart, E. Bennett, and L. McMillen. PixelView: A View-Independet Graphics Rendering Architecture. In M. McCool T. Akenine-Möller, editor, *Graphics Hardware 2004*, 2004.

[9] M. Meïsner, U. Kanus, and W. Straßer. A PCI-Card for Real-Time Volmue Rendering. In *Eurographics Workshop on Graphics Hardware*, pages 61–67, 1998.

[10] Chistiaan Gribble, Steven Parker, and Charles Hansen. A Preliminary Evaluation of the Silicon Graphics Onyx4 UltimateVision Visualization System for Large-Scale Parallel Volmue Rendering. Technical report, University of Utah, School of Computing, Jan 2004.

[11] David E. DeMarle, Chistiaan Gribble, Solomon Boulos, and Steven Parker. Memory Sharing for Interactive Ray-Tracing on Clusters. *Journal of Parallel and Distributed Computing*, 2005.

[12] David E. DeMarle, Christiaan Gribble, and Steven Parker. Memory-Savvy Distributed Interactive Ray-Tracing. In *Eurographics Symposium on Parallel Graphics and Vsualization*, 2004.

[13] David E. DeMarle, Steven Parker, Mark Hartner, Christiaan Gribble, and Charles D. Hansen. Distributed Interactive Ray-Tracing for Large Volume Visualization. In *IEEE Symposium on Parallel and Large-Data Visualization and Graphics*, pages 87–94, 2003.

[14] Adam Moerschell and John D. Owens. Distributed Texture Memory in a Multi-GPU Environment. In *Graphics Hardware 2006*, pages 31–38, Sep 2006.

[15] Steven Molner, Michael Cox, David Ellsworth, and Henry Fuchs. A Sorting Classification of Parallel Rendering. *Computer Graphics and Applications, IEEE*, 14(4):23–32, Jul 1994.

[16] Top500. http://clusters.top500.org, 2004.

[17] Dolphin. http://www.dolphinics.com, 2004.

[18] Randy Pausch, Thomas Crea, and Matthew Conway. A Literature Survey for Virtual Environments: Military Flight Simulator Visual Systems and Simulator Sickness. *Presence: Teleoperators and Virtual Environments*, 1(3):344–363, 1992.

[19] Dolphin News. http://www.dolphinics.com/news/-2000/june020-2000.html, 2000.

[20] Michael Manzke and Brian Coghlan. Non-Intrusive Deep Tracing of SCI Interconnect Traffic. In Wolfgang Karl and Geir Horn, editors, *Conference Proceedings of SCI Europe '99*, pages 53–58. ESPRIT Project 'SCI-Europe' (EP25257) and ESPRIT Working Group 'SCI-WG'(EP22582), Sep 1999.

[21] Michael Manzke, Stuart Kenny, Brian Coghlan, and Olav Lysne. Tuning and Verification of Simulation Models for High Speed Interconnection. In *PDPTA 2001)*, Jun 2002.

[22] Dolphin. http://www.dolphinics.com/-products/hardware/lc3.html, 2005.

[23] ARM Limited. *AMBA Specification, Rev. 2.0*, May 1999.

[24] Jorgen Norendal and Kurt Tjemsland. TLE Version 2 Description and Test Report. Deliverable 25257, SINTEF, March 2001. Work package: WP2.

[25] Eoin Creedon, Ross Brennan, and Michael Manzke. Towards a Scalable Field Programmable Gate Array Cluster for Interactive Parallel Ray-Tracing. In *Eurographics Ireland Workshop*, Oct 2006.