4^{th} Workshop in Virtual Reality Interactions and Physical Simulation "VRIPHYS" (2007)
J. Dingliana, F. Ganovelli (Editors)

Hardware Accelerated Broad Phase Collision Detection for Realtime Simulations

Muiris Woulfe, John Dingliana and Michael Manzke

Graphics, Vision and Visualisation Group (GV2), Department of Computer Science, Trinity College Dublin, Ireland

Abstract

Broad phase collision detection is a vital task in most interactive simulations, but it remains computationally expensive and is frequently an impediment to efficient implementation of realtime graphics applications. To overcome this hurdle, we propose a novel microarchitecture for performing broad phase collision detection using Axis-Aligned Bounding Boxes (AABBs), which exploits the parallelism available in the algorithms. We have implemented our microarchitecture on a Field-Programmable Gate Array (FPGA) and our results show that this implementation is capable of achieving an acceleration of up to 1.5× over the broad phase component of the SOLID collision detection library, when considering the communication overhead between the CPU and the FPGA. Our results further indicate that significantly higher accelerations are achievable using a more sophisticated FPGA or by implementing our microarchitecture on an Application-Specific Integrated Circuit (ASIC).

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Hardware Architecture I.3.5 [Computer Graphics]: Physically based modeling I.3.6 [Computer Graphics]: Graphics data structures and data types

Keywords: [Broad phase collision detection] [Axis-Aligned Bounding Box (AABB)] [Field-Programmable Gate Array (FPGA)]

1. Introduction

Collision detection is a fundamental component of interactive graphics applications as it determines whether simulated objects are in contact. Despite its fundamentality, collision detection remains a challenge and although numerous algorithms and bounding volumes have been proposed, it remains computationally expensive.

Many algorithms have been successfully accelerated by implementing them on a hardware coprocessor, as it is often possible to create hardware adapted to the needs of a particular algorithm. Coprocessors also provide additional CPU time for the execution of the algorithms remaining on the CPU. Two Integrated Circuits (ICs) potentially suitable for use as hardware coprocessors are Field-Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuits (ASICs).

FPGAs can be dynamically reprogrammed to perform whatever functionality is currently desired, thereby implementing a wide variety of microarchitectures over time. FPGAs consequently eliminate the need for multiple expensive ICs. In contrast, ASICs are hardware coprocessors customised for a particular use rather than general-purpose use. They permit more complex designs than FPGAs and can typically execute the same microarchitecture at a higher clock speed. On the downside, their functionality is fixed and cannot be adapted to the requirements of a particular application.

In this paper, we present a novel microarchitecture for performing broad phase collision detection in hardware. Our initial implementation was prototyped on an FPGA but the proposed microarchitecture is equally amenable to efficient ASIC implementation.

The remainder of this paper is organised as follows: Section 2 describes how we determined that it was feasible to accelerate broad phase collision detection by means of a microarchitecture; Section 3 highlights research related to the contribution presented in this paper; Section 4 outlines our microarchitecture for performing broad phase collision detection; Section 5 describes a test scenario we constructed; Section 6 highlights the performance differential between our implementation and the broad phase component of the SOLID collision detection library, along with some additional results; Section 7 concludes this paper and suggests some future work.

2. Motivation

Broad phase collision detection often forms a significant load in physics simulations. Accordingly, we ascertained that it would be desirable to accelerate these computations. After analysing the available broad phase algorithms, we discovered that some offered significant scope for parallelism, making them amenable to hardware coprocessor implementation.

We determined that broad phase collision detection is compute bound, making it possible to perform significant computations on large quantities of data simultaneously. For every n objects communicated, up to $3n(n-1)$ floating-point operations can be performed. This serves to lessen the impact of the communication overhead, as we would not need to communicate small quantities of data between the CPU and the coprocessor on a regular basis.

We further established that broad phase collision detection could benefit from access to substantial low latency, high bandwidth memory resources. A microarchitecture can have extensive quantities of internal dual port RAMs that can be accessed in a single clock cycle, which is of benefit to our broad phase computations.

Based on these analyses, we concluded that it would be desirable to create broad phase collision detection hardware coprocessors. Such coprocessors would offer acceleration and leave the CPU free to perform other computations.

3. Related work

The literature discusses myriad broad phase collision detection bounding volumes, which enclose complex shapes within simpler geometry. One of the most straightforward is the Axis-Aligned Bounding Box (AABB), a cuboid whose lines are parallel to the coordinate axes. Oriented Bounding Boxes (OBBs) are rotatable AABBs and they may offer better performance for certain applications. These can be generalised to Discrete Oriented Polytopes (k-DOPs), which are polyhedra of k sides.

The traditional method for comparing a list of objects for collision is the $\mathcal{O}(n^2)$ brute force method, which performs $\frac{n}{2}(n-1)$ pairwise collision detection comparisons. Alternatively, Baraff's "sweep and prune" (or "sort and sweep") [Bar92, CLMP95] algorithm improves the brute force implementation by considering temporal coherence between frames. In essence, the algorithm assumes that insertion sort can be used to efficiently update the list of object endpoints. The algorithm operates in expected $\mathcal{O}(n \log n)$ time, but it can require up to $\mathcal{O}(n^2)$ time if the degree of coherence is low. Finally, spatial subdivision divides the simulated space into cells. Objects within a cell can only collide with other objects in the same cell, reducing the number of intersection tests that need to be performed. Uniform grids, which split space into cells of equal size, are the simplest spatial subdivision method. Hierarchical grids improve this scheme for objects of varying size by adapting the cells to the size of the objects they enclose. More sophisticated schemes based around tree structures are also possible. Examples include octrees, which recursively partition space into eight uniform cells, and k-d trees [Ben75,FBF77], which recursively partition space into two uniform cells by selecting the most appropriate partitioning axis at each cycle. Binary Space Partitioning (BSP) trees [FKN80] generalise k-d trees by allowing the partition to occur along any arbitrary axis and to create cells of unequal size.

Currently, the bulk of research focusing on offloading collision detection from the CPU utilises GPU shaders. These implementations were traditionally based on imagespace techniques. The algorithm proposed by Myszkowski, Okunev and Kunii [MOK95], along with most early algorithms, is essentially limited to convex objects. Later algorithms, such as RECODE [BWS98] and CInDeR [KP03], process unions of convex objects. CULLIDE [GRLM03] can use general polygon soups, but the implementation uses a hybrid approach with certain computations remaining on the CPU. RCULLIDE [GLM04] improves on CULLIDE, overcoming limited viewport resolutions by computing the Minkowski sum of each primitive with a sphere, to perform reliable 2.5D overlap tests. Despite the performance advantages achieved, imagespace approaches work with reduced precision [RLMK05] and more recently, research has focused on using GPUs to directly implement the mathematical equations comprising collision detection algorithms. Carr, Hall and Hart [CHH02] and Purcell et al. [PBMH02] discuss the implementation of ray-triangle intersection tests using these techniques. Alternatively, Havok FX [Hav] is a commercially available physics engine plug-in that simulates background effects physics. Despite the advantages provided by developments in GPU collision detection, all of these methods typically compete for GPU time with the rendering instructions. To relieve the congestion, a second GPU could be dedicated to collision detection computations, but most of its resources would be unutilised.

In the field of FPGA collision detection, Raabe et al. [RHAZ06] implemented a complete collision detection microarchitecture. Their broad phase algorithm checks k-DOPs for collision using a novel algorithm based on the Separating Axis Theorem (SAT). Fixed-point arithmetic is used to reduce the size of the microarchitecture, which is designed to be space-efficient, so other algorithms can be implemented on the same FPGA alongside the presented microarchitecture. The authors achieve an acceleration of a factor of four over an equivalent software implementation for a static scene, but this figure excludes the communication overhead required for any practical application. Since our results indicate a significant communication overhead, it is unlikely that this figure can be attained in practice. The microarchitecture can also simulate dynamic scenes although this is unsupported by the current API, and results for dynamic scenes remain unavailable. Alternatively, Atay, Lockwood and Bayazit [ALB05] describe a microarchitecture based on Möller's triangle-triangle intersection test [Möl97], which allows for exact collision detection in an environment described entirely by triangles. However, their target application is motion planning for robots [AB06] and this microarchitecture is unable to perform collision detection in realtime for typical physics simulations.

Few collision detection ASICs have been designed due to the significant time and capital investments required. The sole example is AGEIA PhysX [AGE], a commercially available Physics Processing Unit (PPU), that reportedly accelerates both physics and collision detection. Implementation details of the microarchitecture have not been published in academic fora, although patents [DHS*05a, DHS*05b, DHS*05c, ZTSM05a, TZS05, ZTSM05b, MBST05, BS06, BDH06, MFSD07, Bor07] reveal some details. It further remains unclear how much of the collision detection is performed on the ASIC and how much is performed on the CPU.

FPGAs have also been used to successfully accelerate a plurality of graphics algorithms. In particular, Schmittler et al. [SWW*04] present a novel design for performing realtime raytracing on an FPGA. Wei et al. [WCF*98] simulated a distributed frame buffer using an FPGA while Wetekam et al. [WSKW05] used FPGAs to perform multi-resolution volume rendering.

A substantial body of research indicates that floating-point computations can be accelerated by FPGAs. In 2004, Underwood [Und04] concluded that peak FPGA floating-point performance is increasing at a rate of up to 5× every two years, compared to 2× every eighteen months on CPUs. Also in 2004, Govindu et al. [GZCP04] demonstrated an FPGA-based matrix multiplication algorithm that was capable of achieving up to 15 GFLOPS, significantly faster than the available CPU technology. This research clearly demonstrates that although FPGAs were once deemed unsuitable for floating-point operations, they have recently become a viable platform for applications reliant on floating-point numbers.

We have previously presented a work in progress poster of an initial implementation of our microarchitecture [WDM07]. This poster focused on our comparator module, which is described in Section 4.3, and provided some preliminary performance data.

4. Microarchitecture

Our custom broad phase collision detection microarchitecture consists of four modules, connected as illustrated in Figure 1. The four modules are the updater, the comparator, the collision emitter and the CPU–IC communication interface. These modules will be described in the subsequent sections.

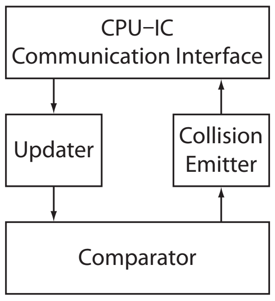


Figure 1: Interconnection of the modules in our microarchitecture.

Our microarchitecture accommodates scalability, in order to adapt to differing IC sizes. Therefore, for small ICs a small number of collision detection tests can be performed in parallel while on a larger IC a large number of tests can be performed in parallel. The diagrams in the following sections illustrate this scalability.

Our microarchitecture uses single precision floating-point arithmetic [IEE85], as our empirical observations suggested that this offered sufficient precision. The use of a standard floating-point precision also simplifies communication with the CPU, as no conversion hardware or routines are required.

The microarchitecture operates in two general phases: the initialisation phase is the first phase and loads the initial data into the microarchitecture; the subsequent updating phases update the stored data and perform broad phase collision detection.

4.1. Bounding volume and algorithm

We decided to use AABBs as bounding volumes. The simplicity of AABBs means fewer large floating-point cores are required, which enables more collisions to be detected in parallel. Moreover, AABBs have been successfully used by the I-COLLIDE [CLMP95] and SOLID [van01] collision detection libraries.

We further decided to use the brute force algorithm to check the AABBs for collision. Although the brute force algorithm may initially appear disadvantageous, we selected it because it offers significant scope for parallelism, making it amenable to efficient hardware implementation. Additionally, it allowed us to quickly and effectively test the appropriateness of offloading broad phase collision detection to a hardware coprocessor. In the future, we intend to look at possibilities for adapting our microarchitecture to execute more sophisticated algorithms such as spatial subdivision.

4.2. Updater

The updater module, illustrated in Figure 2, manages the data describing each AABB.

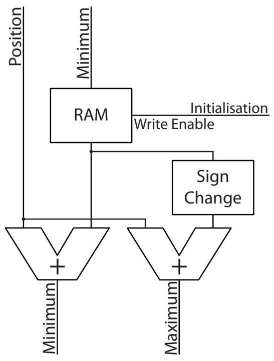


Figure 2: The microarchitecture of the updater module.

During the initialisation phase, the software application centres each simulated object at the origin of the three coordinate axes and computes an AABB for each object. It provides the AABBs as three data (minimum x, minimum y, minimum z), and the updater module stores these values in on-chip RAM for subsequent use. Only the minimum values are required since the AABBs are centred at the origin and the maximum values can be calculated by switching the sign of the corresponding minimum value.

Subsequently, during the updating phase, the current centre of each simulated object is provided to the updater module. These new centres are added to the stored minima and computed maxima to calculate the current minima and maxima. This data is provided to the next module, the comparator.

4.3. Comparator

The comparator module is the principal module in the microarchitecture. It is responsible for performing the appropriate collision detection tests between pairs of AABBs and reporting their indices if they are colliding. The sequential algorithm equivalent to our comparator module implementation is shown in Algorithm 1.

The microarchitecture is illustrated from two different perspectives. Figure 3 shows how a single minimum and maximum can be combined to form the entire module, while Figure 4 shows how a number of AABBs are processed for collision. Figure 3 demonstrates the repetitious nature of the microarchitecture, while Figure 4 views the module from a higher-level perspective, from the view of an AABB.

```
for $i \leftarrow 1$ to $n-1$ do
    for $j \leftarrow i+1$ to $n$ do
        collision $\leftarrow (\mathbf{x}_i^{max} \geq \mathbf{x}_j^{min}) \wedge (\mathbf{x}_i^{min} \leq \mathbf{x}_j^{max})$
                  $\wedge (\mathbf{y}_i^{max} \geq \mathbf{y}_j^{min}) \wedge (\mathbf{y}_i^{min} \leq \mathbf{y}_j^{max})$
                  $\wedge (\mathbf{z}_i^{max} \geq \mathbf{z}_j^{min}) \wedge (\mathbf{z}_i^{min} \leq \mathbf{z}_j^{max})$
        if collision then
            result $\leftarrow$ result $\cup \langle i, j \rangle$
        end if
    end for
end for
```

Algorithm 1: The sequential brute force algorithm for detecting AABB collisions.

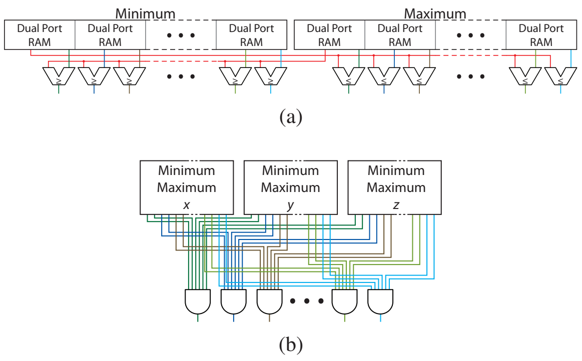


Figure 3: (a) The relevant components for processing a single coordinate axis. (b) The interconnection of these components to form the complete comparator module.

The comparator module begins with a $2m$ width memory. This can be constructed using m dual port RAMs with the same data replicated across each RAM, logically creating a $2m$ port RAM. This $2m$ port RAM allows $2m$ data to be outputted in a single clock cycle, instead of the typical two data. $2m$ data facilitate a greater degree of parallelism in the comparison logic, allowing for greater acceleration. In total, six $2m$ width RAMs are required for storing the minimum and maximum x, y and z data.

When writing to the RAMs, the same address is used for each of the m RAMs so that a single datum is replicated across each RAM. Subsequently, when reading from the RAMs, a more complex scheme, exemplified in Table 1, is used to ensure each of the necessary comparisons take place. In this scheme, the first address input is set to the initial address and the remaining $2m-1$ address inputs are set to the $2m-1$ successive addresses. In the subsequent cycles, the first address input is kept constant while the remaining $2m-1$ address inputs are each incremented by $2m-1$ until one of these address inputs reaches the maximum address used in the system. When this occurs, the first address is incremented by 1 and the remaining $2m-1$ address inputs are set to the $2m-1$ addresses subsequent to the new first address. The process continues until the first address

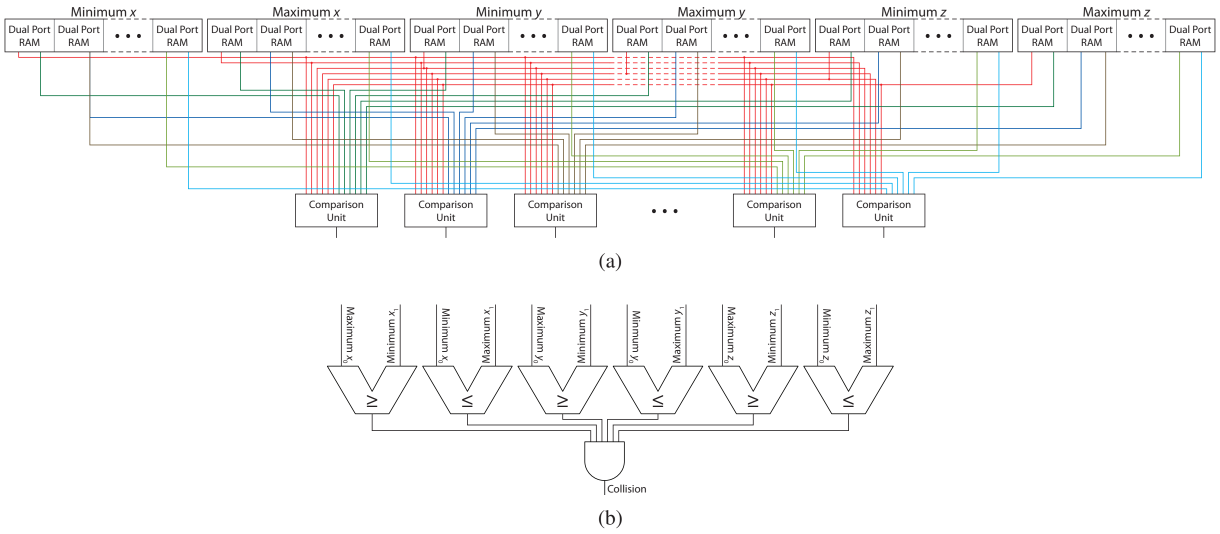


Figure 4: (a) The comparator module. (b) The implementation of each comparison unit.

input reaches one less than the maximum address and the remaining $2m-1$ address inputs reach the maximum address. This addressing scheme permits addresses greater than the maximum address to be accessed (these are illustrated as '–' in Table 1), but these are removed by checking the supplied AABB index against the maximum address.

On-chip RAM	0		1		2		3	
Port	A	B	A	B	A	B	A	B
Input	0	1	2	3	4	5	6	7
	0	1	2	3	4	5	6	7
	0	8	9	10	–	–	–	–
	1	2	3	4	5	6	7	8
	1	9	10	–	–	–	–	–
	2	3	4	5	6	7	8	9
	2	10	–	–	–	–	–	–
	3	4	5	6	7	8	9	10
	4	5	6	7	8	9	10	–
	5	6	7	8	9	10	–	–
	6	7	8	9	10	–	–	–
	7	8	9	10	–	–	–	–
	8	9	10	–	–	–	–	–
	9	10	–	–	–	–	–	–

Table 1: An example of the addresses supplied to the RAMs, where $m = 4$ and the maximum address is 10.

The comparison in Algorithm 1 is also implemented using six floating-point cores with each performing one comparison. The result from the six cores is ANDed to determine whether a collision occurred. If a collision occurred, the indices of the two colliding AABBs are passed to the next module, the collision emitter.

It is in the comparison operation that our microarchitecture uses techniques to surpass the performance of CPUs. Parallelism has been successfully used to accelerate myriad applications, so we decided to perform all six comparisons in Algorithm 1 in parallel. Moreover, we perform these six comparisons for $2m-1$ pairs of AABBs in parallel, facilitated by our $2m$ width RAM.

4.4. Collision emitter

The collision emitter module, illustrated in Figure 5, converts the parallel collision data supplied by the comparator module into a serial stream suitable for writing to the RAM used for communication with the CPU. Each collision is specified by two indices representing two colliding AABBs. Up to $2m-1$ parallel collisions are written into $4m-2$ FIFOs simultaneously, which are then read sequentially using a round robin scheduling algorithm.

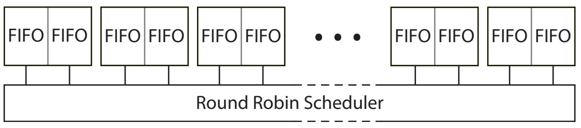


Figure 5: The microarchitecture of the collision emitter module.

This collision data must be serialised, as it would not be possible to communicate parallel collisions to the CPU. Despite the serialisation, this module does not appear to be a substantial bottleneck as typically less than one collision is emitted per clock cycle.

4.5. CPU–IC communication interface

The CPU–IC communication interface is responsible for managing the flow of data between the CPU executing the appropriate software application and the IC implementing the microarchitecture described. This currently comprises external RAM hosted on the coprocessor board, which can be accessed by both the CPU and IC, along with the infrastructure required to read and write data to and from this RAM. A number of equivalent implementations are also possible and the interface can be easily tuned to the requirements of the IC in use.

During the initialisation phase, the CPU writes the origin-centred AABB minima into the RAM. The IC reads this data and performs the necessary computation. Subsequently, during the updating phase, the CPU writes the new centre positions of each AABB. The IC reads this data and performs the required updating and collision detection operations, before writing the indices of the colliding AABBs into the RAM, where the CPU accesses them. Externally accessible registers allow the CPU to inform the IC of the quantity of AABBs and allow the IC to inform the CPU of how many collisions occurred.

4.5.1. FPGA

We adapted our microarchitecture for implementation on an FPGA. The adapted microarchitecture uses six parallel Synchronous Static Random Access Memories (SSRAMs) for communication between the CPU and the FPGA, allowing six data to be read or written on each clock cycle. The FPGA board hosting the SSRAMs is discussed in greater detail in Section 6.

During the initialisation phase, the CPU writes the x, y and z of the origin-centred AABB minima to a single address across three SSRAMs, permitting the FPGA to read the x, y and z data in parallel. In the updating phase, the CPU writes the x, y and z of the current centre of the AABBs to a single address across three SSRAMs. In both cases, the CPU writes to n addresses for n AABBs, resulting in the transfer of $3n$ data. In the updating phase, after performing the collision detection computations, the FPGA writes the two AABB indices specifying each collision to an SSRAM, writing to n addresses for n collisions.

Our FPGA implementation is currently limited to a maximum of 512 AABBs due to the depth of the FPGA's on-chip Block RAMs, although the latest FPGAs would extend this limit to 1024 AABBs. We can surpass this limit with some minor adjustments to our microarchitecture, since the external SSRAMs are substantially larger and could be used to contain up to 524,288 AABBs. If we deem 524,288 to be too limiting, we can also exceed this restriction by streaming AABBs between the CPU and FPGA.

5. Test scenario

In addition to constructing the previously described microarchitecture, we constructed a sample software application that can either execute in software or interact with the microarchitecture to perform a physically based simulation. This simulation consists of a central program loop executing broad phase collision detection, narrow phase collision detection, collision response, physics simulation and graphical display sequentially, as illustrated in Figure 6.

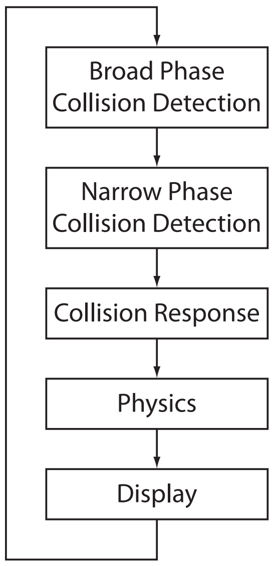

Broad Phase Collision Detection
Narrow Phase Collision Detection
Collision Response
Physics
Display

Figure 6: The software application's central program loop.

Our test scenario simulates a closed box initialised with a variable number of spheres. The box is of height 400 m and the width and depth are calculated so that on initialisation there exists a gap of 10 m between each sphere of radius 25 m. Each sphere is initialised with a unique initial linear momentum in the range $(-40\text{ m/s}, -40\text{ m/s}, -40\text{ m/s})$ to $(40\text{ m/s}, 40\text{ m/s}, 40\text{ m/s})$. Almost all of the resultant velocities stay within the range $(-15\text{ m/s}, -15\text{ m/s}, -15\text{ m/s})$ to $(15\text{ m/s}, 15\text{ m/s}, 15\text{ m/s})$, so that all objects appear to behave realistically.

During the course of the simulation, the spheres collide with each other and the inner walls of the box. By adjusting the number of spheres and their velocities, the computational load placed on the host computer can be varied, which can be used to determine the impact of collision detection on the overall system. Figure 7 shows a screen capture of the test scenario simulating 500 spheres. In the future, a multitude of alternative test scenarios that effectively utilise our microarchitecture could also be constructed.

	FPGA		SOLID		Acceleration				Communication			Collisions
	$m = 4$	$m = 8$	incl sort	excl sort	$m = 4$	$m = 4$	$m = 8$	$m = 8$		$m = 4$	$m = 8$	
	ms	**ms**	**ms**	**ms**	incl sort	excl sort	incl sort	excl sort	**ms**	**%**	**%**	
50	0.242	0.216	0.043	0.042	0.176	0.175	0.197	0.195	0.178	73.542	82.096	7.47
100	0.289	0.276	0.146	0.139	0.504	0.48	0.528	0.502	0.194	67.304	70.478	12.588
150	0.332	0.289	0.195	0.163	0.588	0.489	0.675	0.562	0.199	59.825	68.701	20.413
200	0.356	0.312	0.318	0.218	0.894	0.612	1.02	0.699	0.208	58.418	66.653	26.703
250	0.419	0.348	0.597	0.353	1.423	0.842	1.713	1.014	0.218	51.958	62.578	35.628
300	0.471	0.368	0.914	0.418	1.943	0.888	2.484	1.135	0.225	47.772	61.064	44.61
350	0.5	0.394	1.431	0.501	2.864	1.002	3.632	1.270	0.235	47.094	59.709	51.416
400	0.569	0.425	2.708	0.575	4.763	1.012	6.377	1.355	0.25	43.935	58.822	60.053
450	0.631	0.464	4.92	0.645	7.802	1.023	10.6	1.39	0.253	40.183	54.594	64.427
500	0.731	0.513	8.325	0.754	11.395	1.032	16.243	1.471	0.27	37.001	52.743	76.079

Table 2: Average execution time of a broad phase collision detection cycle for a variable quantity of AABBs.

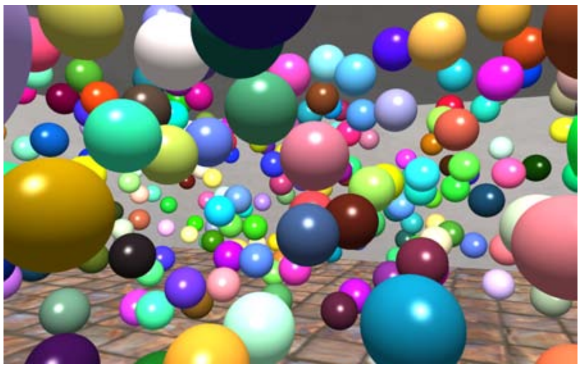


Figure 7: A screen capture of our test scenario simulating 500 spheres.

6. Results

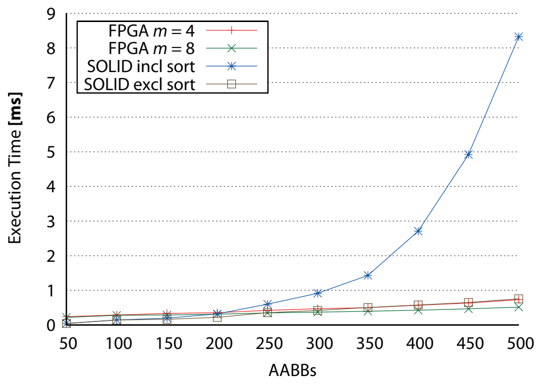
Our test scenario is implemented in C++. It was compiled using Microsoft Visual Studio 2005 and was tested on an Intel Pentium 4 1.8 GHz with 2 GB of RAM. The design was optimised for speed.

Our microarchitecture is implemented in VHDL. For our analyses, we targeted a Xilinx Virtex-II XC2V6000-4 FPGA [Xil], hosted on an Alpha Data ADM-XRC-II board [Alp], which communicates with the CPU via 32 bits of the PCI bus operating at 66 MHz. The board includes six independent 512 KB banks of 36-bit wide SSRAM operating at the FPGA clock speed. The FPGA features 144 18-bit multipliers, 324 KB of Block RAM and 6 million gate equivalents. Xilinx ISE 9.1.03i was used to synthesise, translate, map, place and route the VHDL description of our entire microarchitecture. This enabled our test scenario to perform all broad phase collision detection on the FPGA. The design was optimised for speed.

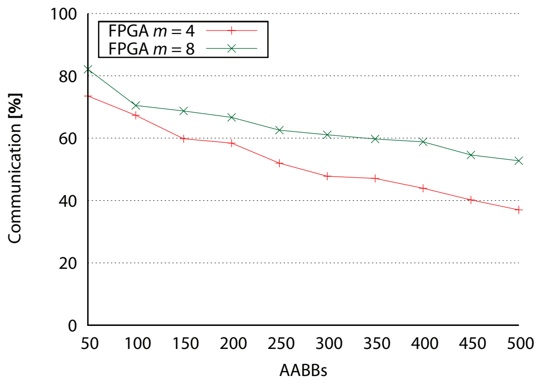
When synthesised with $m = 4$, the microarchitecture achieved a target frequency of 108 MHz and consumed 29% of FPGA slices and 28% of available Block RAM. The peak memory bandwidth achieved by the comparator module's 8 ($2m$) width RAM was 3.219 GB/s. When synthesised with $m = 8$, the microarchitecture also achieved a target frequency of 108 MHz and consumed 41% of FPGA slices and 56% of available Block RAM. The peak memory bandwidth achieved by the comparator module's 16 ($2m$) width RAM was 6.437 GB/s. In both cases, when the microarchitecture reads the data using the scheme outlined in Table 1, the quoted peak memory bandwidth is attained.

To analyse the performance of our microarchitecture, we timed our FPGA implementation against the SOLID 3.5.6 collision detection library. The results of this comparison are provided in Table 2 and plotted in Figure 8. The execution times provided are the time consumed by a single collision detection cycle averaged over 10,000 cycles, recorded using high-resolution hardware performance counters. Only the time consumed by the broad phase computations, excluding the narrow phase collision detection, collision response, physics and graphical display, were recorded. FPGA times include the time consumed while preparing and transferring data to and from the CPU using Direct Memory Access (DMA) transfers. SOLID times both including and excluding sweep and prune's lengthy initial sort computation are presented. Table 2 also quotes the relative performance of the FPGA and SOLID implementations, the communication overhead and the average number of collisions per cycle.

These results indicate that our FPGA implementation decelerates low quantities of AABBs, primarily due to the communication overhead involved. It accelerates 250 AABBs by a factor of 1.014× and it accelerates 500 AABBs by a factor of 1.471×. It is also apparent that the increased parallelism of the $m = 8$ implementation provides a moderate acceleration, but its performance is dampened by the significant and constant communication overhead. Although both the computer and the FPGA used to generate these results are a number of years old, they both date from the same year, making it valid to compare the re-



(a)



(b)

Figure 8: (a) The average execution time of a broad phase collision detection cycle for a variable quantity of AABBs. (b) The communication overhead of the FPGA implementation.

sults of one against the other. Unfortunately, it was impossible to compare our results against the AGEIA PhysX as the API provides access to the broad phase collision detection, narrow phase collision detection, collision response and physics simulation through a simple high level API, which does not permit direct access to the broad phase implementation.

For more than 500 AABBs, we predict a larger acceleration factor, which we deduced from the pattern of acceleration values in Table 2. Further, placing the microarchitecture on a PCI Express FPGA card would significantly lessen the communication overhead, while an ASIC implementation would be of added benefit, as it would enable a higher clock speed coupled with an even greater degree of parallelism.

7. Conclusions and future work

In this paper, we presented a novel microarchitecture for performing broad phase collision detection for interactive simulations. Our FPGA prototype demonstrates that this microarchitecture is capable of achieving an acceleration of up to $1.5\times$ over the broad phase component of the SOLID collision detection library. Moreover, current and future trends suggest that this performance advantage will improve as newer and more advanced hardware technologies become available.

Our implementation and results show that it is possible to successfully implement a microarchitecture for accelerating broad phase collision detection. Additionally, it is feasible to divide a graphical application with realtime constraints between a CPU and coprocessor, with the effect of the communication overhead diminished by the acceleration achieved. An FPGA provides a suitable host for this coprocessor as our microarchitecture's parallelism compensates for the relatively low clock speed of the devices. Alternatively, an ASIC would permit a faster implementation and allow for more flexibility in terms of computational resources.

In the future, we plan to design and implement solutions to overcome the current limitations of our microarchitecture. Our test scenario used spheres that were invariant to rotation, which simplified our application since AABBs did not need to be recomputed whenever a rotation occurred. Future work will involve adding support for more general objects and motions using dynamic AABBs or alternative bounding volumes. We will also modify our architecture so that significantly more than 512 AABBs can be simulated concurrently, and we plan to explore more sophisticated algorithms than brute force to potentially achieve a greater acceleration.

We further intend to adapt our broad phase collision detection microarchitecture to execute on the shared-memory hybrid graphics cluster for visualisation and video processing [MBO*06,BMO*07]. The system consists of a cluster of custom-built Printed Circuit Boards (PCBs), connected via Scalable Coherent Interface (SCI), which provides a high bandwidth, low latency, point-to-point interconnect implementing a Distributed Shared Memory (DSM) architecture. After porting our microarchitecture, all FPGAs in the cluster will be able to perform collision detection in parallel, vastly increasing the parallelism of the system. Furthermore, the SCI interconnect will alleviate the delay incurred by communicating over the PCI bus in a commodity PC. The substantial parallelism and low latency interconnect should serve to considerably increase the performance of our microarchitecture.

Finally, we aim to profile and analyse our hardware accelerated test scenario to establish its limiting components. We will utilise these data to enable us to optimally distribute the required computations across all available resources, including the CPU and hardware coprocessor. By dividing computations in this manner, we hope to achieve maximum acceleration for realtime simulations by means of our hardware coprocessor.

Acknowledgements

This research is supported by the Irish Research Council for Science, Engineering and Technology (IRCSET) funded by the National Development Plan (NDP).

References

[AB06] ATAY N., BAYAZIT B.: A motion planning processor on reconfigurable hardware. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation (ICRA 2006)* (Piscataway, New Jersey, USA, May 2006), Papanikolopoulos N., (Ed.), IEEE Press, pp. 125–132.

[AGE] AGEIA PhysX. http://www.ageia.com/physx/ (Retrieved 25 September 2007).

[ALB05] ATAY N., LOCKWOOD J. W., BAYAZIT B.: A collision detection chip on reconfigurable hardware. In *Proceedings of the Thirteenth Pacific Conference on Computer Graphics and Applications (Pacific Graphics 2005)* (Oct. 2005), Gotsman C., Manocha D., Wu E., (Eds.).

[Alp] Alpha Data. http://www.alpha-data.com/ (Retrieved 25 September 2007).

[Bar92] BARAFF D.: *Dynamic Simulation of Non-Penetrating Rigid Bodies*. PhD thesis, Cornell University, Ithaca, New York, USA, May 1992.

[BDH06] BORDES J. P., DAVIS C., HEDGE M.: System with PPU/GPU architecture, May 2006. (International application number: PCT/US2005/040007).

[Ben75] BENTLEY J. L.: Multidimensional binary search trees used for associative searching. *Communications of the ACM 18*, 9 (Sept. 1975), 509–517.

[BMO*07] BRENNAN R., MANZKE M., O'CONOR K., DINGLIANA J., O'SULLIVAN C.: A scalable and reconfigurable shared-memory graphics cluster architecture. In *Proceedings of the 2007 International Conference on Engineering of Reconfigurable Systems & Algorithms (ESRA 2007)* (Long Island City, New York, USA, June 2007), Plaks T. P., (Ed.), CSREA.

[Bor07] BORDES J. P.: System and method providing variable complexity in a physics simulation, Mar. 2007. (International application number: PCT/US2006/028164).

[BS06] BORDES J. P., SEQUEIRA D.: Asynchronous and parallel execution by physics processing unit, May 2006. (International application number: PCT/US2005/040006).

[BWS98] BACIU G., WONG W. S.-K., SUN H.: RE-CODE: An image-based collision detection algorithm. In *Proceedings of the Sixth Pacific Conference on Computer Graphics and Applications (Pacific Graphics '98)* (Los Alamitos, California, USA, Oct. 1998), Teh H. C., Shinagawa Y., Patrikalakis N. M., (Eds.), IEEE Computer Society, pp. 125–133.

[CHH02] CARR N. A., HALL J. D., HART J. C.: The ray engine. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Conference on Graphics Hardware* (Aire-la-Ville, Switzerland, Sept. 2002), Ertl T., Heidrich W., Doggett M., (Eds.), Eurographics Association, pp. 37–46.

[CLMP95] COHEN J. D., LIN M. C., MANOCHA D., PONAMGI M. K.: I-COLLIDE: An interactive and exact collision detection system for large-scale environments. In *Proceedings of the 1995 Symposium on Interactive 3D Graphics (SI3D '95)* (New York, USA, Apr. 1995), ACM Press, pp. 189–196.

[DHS*05a] DAVIS C., HEDGE M., SCHMID O. A., MAHER M., BORDES J. P.: Method for providing physics simulation data, Apr. 2005. (International application number: PCT/US2004/030687).

[DHS*05b] DAVIS C., HEDGE M., SCHMID O. A., MAHER M., BORDES J. P.: Physics processing unit, Apr. 2005. (International application number: PCT/US2004/030686).

[DHS*05c] DAVIS C., HEDGE M., SCHMID O. A., MAHER M., BORDES J. P.: System incorporating physics processing unit, Apr. 2005. (International application number: PCT/US2004/030689).

[FBF77] FRIEDMAN J. H., BENTLEY J. L., FINKEL R. A.: An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software 3*, 3 (Sept. 1977), 209–226.

[FKN80] FUCHS H., KEDEM Z. M., NAYLOR B. F.: On visible surface generation by a priori tree structures. In *Proceedings of the 7th International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '80)* (New York, USA, July 1980), Thomas J. J., (Ed.), ACM Press, pp. 124–133.

[GLM04] GOVINDARAJU N. K., LIN M. C., MANOCHA D.: Fast and reliable collision culling using graphics hardware. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST 2004)* (New York, USA, Nov. 2004), Lau R. W. H., Baciu G., (Eds.), ACM Press, pp. 2–9.

[GRLM03] GOVINDARAJU N. K., REDON S., LIN M. C., MANOCHA D.: CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Conference on Graphics Hardware* (Aire-la-Ville, Switzerland, July 2003), Doggett M., Heidrich W., Mark W., Schilling A., (Eds.), Eurographics Association, pp. 25–32.

[GZCP04] GOVINDU G., ZHAO L., CHOI S., PRASANNA V.: Analysis of high-performance floating-point arithmetic on FPGAs. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS 2004)* (Los Alamitos, California, USA, Apr. 2004), Vernalde S., (Ed.), IEEE Computer Society, pp. 149–156.

[Hav] Havok FX. http://www.havok.com/content/view/72/57/ (Retrieved 25 September 2007).

[IEE85] IEEE std 754-1985: IEEE standard for binary floating-point arithmetic, Mar. 1985.

[KP03] KNOTT D., PAI D. K.: CInDeR: Collision and Interference Detection in Real-time using graphics hardware. In *Proceedings of Graphics Interface 2003* (Natick, Massachusetts, Canada, June 2003), Möller T., Ware C., (Eds.), A K Peters, pp. 73–80.

[MBO*06] MANZKE M., BRENNAN R., O'CONOR K., DINGLIANA J., O'SULLIVAN C.: A scalable and reconfigurable shared-memory graphics architecture. In *The 33th International Conference and Exhibition on Computer Graphics and Interactive Techniques (SIGGRAPH 2006)* (New York, USA, July–Aug. 2006), Pfister H., (Ed.), ACM Press.

[MBST05] MAHER M., BORDES J. P., SEQUEIRA D., TONGE R.: Physics processing unit instruction set architecture, Nov. 2005. (International application number: PCT/US2004/030690).

[MFSD07] MÜLLER-FISCHER M. H., SCHIRM S. D., DUTHALER S. F.: Method of generating surface defined by boundary of three-dimensional point cloud, Feb. 2007. (International application number: PCT/US2006/028161).

[MOK95] MYSZKOWSKI K., OKUNEV O. G., KUNII T. L.: Fast collision detection between complex solids using rasterizing graphics hardware. *The Visual Computer 11*, 9 (Nov. 1995), 497–511.

[Möl97] MÖLLER T.: A fast triangle-triangle intersection test. *Journal of Graphics Tools (JGT) 2*, 2 (1997), 25–30.

[PBMH02] PURCELL T. J., BUCK I., MARK W. R., HANRAHAN P.: Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2002) 21*, 3 (July 2002), 703–712.

[RHAZ06] RAABE A., HOCHGÜRTEL S., ANLAUF J., ZACHMANN G.: Space-efficient FPGA-accelerated collision detection for virtual prototyping. In *Proceedings of the 2006 Design, Automation and Test in Europe Conference and Exhibition (DATE 2006)* (Leuven, Belgium, Mar. 2006), European Design and Automation Association, pp. 206–211.

[RLMK05] REDON S., LIN M. C., MANOCHA D., KIM Y. J.: Fast continuous collision detection for articulated models. *Journal of Computing and Information Science in Engineering 5*, 2 (June 2005), 126–137.

[SWW*04] SCHMITTLER J., WOOP S., WAGNER D., PAUL W. J., SLUSALLEK P.: Realtime ray tracing of dynamic scenes on an FPGA chip. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Conference on Graphics Hardware* (Aire-la-Ville, Switzerland, Aug. 2004), Akenine-Möller T., McCool M., (Eds.), Eurographics Association, pp. 95–106.

[TZS05] TONGE R., ZHANG L., SEQUEIRA D.: Method and program solving LCPs for rigid body dynamics, Aug. 2005. (International application number: PCT/US2004/030691).

[Und04] UNDERWOOD K.: FPGAs vs. CPUs: Trends in peak floating-point performance. In *Proceedings of the 12th ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA 2004)* (New York, USA, Feb. 2004), Schmit H., (Ed.), ACM Press, pp. 171–180.

[van01] VAN DEN BERGEN G.: Proximity queries and penetration depth computation on 3D game objects. In *Game Developers Conference 2001 Proceedings (GDC 2001)* (Manhasset, New York, USA, Apr. 2001), CMP Media.

[WCF*98] WEI B., CLARK D. W., FELTEN E. W., LI K., STOLL G.: Performance issues of a distributed frame buffer on a multicomputer. In *Proceedings of the 1998 ACM SIGGRAPH/Eurographics Conference on Graphics Hardware* (New York, USA, Aug.–Sept. 1998), Spencer S. N., (Ed.), ACM Press, pp. 87–96.

[WDM07] WOULFE M., DINGLIANA J., MANZKE M.: Hardware accelerated broad phase collision detection. In *The 34th International Conference and Exhibition on Computer Graphics and Interactive Techniques (SIGGRAPH 2007)* (New York, USA, Aug. 2007), Alexa M., Finkelstein A., (Eds.), ACM SIGGRAPH.

[WSKW05] WETEKAM G., STANEKER D., KANUS U., WAND M.: A hardware architecture for multi-resolution volume rendering. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Conference on Graphics Hardware* (New York, USA, July 2005), Meißner M., Schneider B.-O., (Eds.), ACM Press, pp. 45–51.

[Xil] Xilinx. http://www.xilinx.com/ (Retrieved 25 September 2007).

[ZTSM05a] ZHANG L., TONGE R., SEQUEIRA D., MAHER M.: Method of operation for parallel LCP solver, Aug. 2005. (International application number: PCT/US2004/030688).

[ZTSM05b] ZHANG L., TONGE R., SEQUEIRA D., MAHER M.: Parallel LCP solver and system incorporating same, Aug. 2005. (International application number: PCT/US2004/030692).