Woulfe and Manzke *EURASIP Journal on Embedded Systems* (2017) 2017:1
DOI 10.1186/s13639-016-0037-7

EURASIP Journal on Embedded Systems

RESEARCH Open Access

A hybrid fixed-function and microprocessor solution for high-throughput broad-phase collision detection

Muiris Woulfe* and Michael Manzke

Abstract

We present a hybrid system spanning a fixed-function microarchitecture and a general-purpose microprocessor, designed to amplify the throughput and decrease the power dissipation of collision detection relative to what can be achieved using CPUs or GPUs alone. The primary component is one of the two novel microarchitectures designed to perform the principal elements of broad-phase collision detection. Both microarchitectures consist of pipelines comprising a plurality of memories, which rearrange the input into a format that maximises parallelism and bandwidth. The two microarchitectures are combined with the remainder of the system through an original method for sharing data between a ray tracer and the collision-detection microarchitectures to minimise data structure construction costs. We effectively demonstrate our system using several benchmarks of varying object counts. These benchmarks reveal that, for over one million objects, our design achieves an acceleration of 812× relative to a CPU and an acceleration of 161× relative to a GPU. We also achieve energy efficiencies that enable the mitigation of silicon power-density challenges, while making the design amenable to both mobile and wearable computing devices.

Keywords: Broad phase, Collision detection, Fixed-function microarchitecture, Microprocessor, Hybrid system, Energy efficiency

1 Introduction

As technology progresses, increasingly greater realism is demanded by the consumers of real-time graphics applications. Collision detection is an important factor in achieving this realism. It determines if simulated objects are intersecting, and, in cooperation with collision response, it maintains realism by preventing objects from interpenetrating. Collision detection is found in computer games, animation, robotics and computer-aided design (CAD). An improvement in collision detection will benefit myriad applications.

Despite decades of research, collision detection remains a fundamental problem. It can form a computational bottleneck in many applications. Interactive applications are particularly challenging as they demand a frame rate of at least 30 fps to ensure the illusion of visual continuity. Moreover, the inter-frame durations must be sufficient to execute the entire program loop, which potentially comprises input processing, collision detection, collision response, physics, AI, audio and rendering. The classic solution is to trade accuracy for speed. This trade-off is undesirable for most applications, and it is particularly problematic for robotics and CAD. Additional research is necessary to find sufficient throughput enhancements.

Algorithms can be executed on fixed-function microarchitectures on platforms such as application-specific integrated circuits (ASICs) or on general-purpose microprocessors such as CPUs and GPUs. Microarchitectures sacrifice programmability to dissipate less power and exhibit superior throughput. These advantages result from providing the designer with complete control over component layout and from eliminating the overhead of executing instructions. As many graphics applications require the recurrent execution of algorithms at interactive frame rates, these algorithms are good candidates for microarchitectures, providing they are utilised sufficiently and do not require programmability. GPU rasterisation is a good example of an effective microarchitecture.

*Correspondence: woulfem@tcd.ie
Graphics, Vision and Visualisation Group (GV2), School of Computer Science and Statistics, Trinity College Dublin, Dublin, Ireland

Recent articles on the topic of integrated circuit (IC) power consumption have demonstrated that future ICs will require additional functionality to be implemented as microarchitectures. A power dissipation problem is evident in current multicore architectures. Native transistor switching speeds continue to double every two process generations, while processor frequencies are not increasing substantially. This serves to reduce the amount of utilisation necessary to justify adding custom microarchitectures. Their addition is further justified through the current desire for mobile and wearable computing devices, which demand energy efficiency to maximise finite battery lifespans.

We identify collision detection as an algorithm that is computationally expensive and satisfies the utilisation requirement for its implementation as a microarchitecture. We specifically select the broad phase due to its parallelisability, its compute-bound nature and its need for minimal control logic. Two alternative microarchitectures are proposed: one focuses on minimising resource consumption while the other supports greater object quantities. Both use pipelines comprising a plurality of memories that rearrange the input into a format maximising parallelism and bandwidth. To increase the object counts supported and to improve the computational complexity, we propose a hybrid solution that combines these microarchitectures with a spatial-partitioning stage on a CPU or GPU. We further propose reusing the hierarchies created by a ray tracer to minimise construction costs. For 1,024,000 objects, this system achieves an acceleration of 812× relative to a CPU and an acceleration of 161× relative to a GPU, while maintaining energy efficiency.

This article makes the following contributions:

- Two fixed-function microarchitectures for performing broad-phase collision detection that offer significant throughput and power advantages relative to CPU and GPU equivalents
- A novel technique for combining these microarchitectures with ray-tracing data structures hosted on a general-purpose microprocessor
- A hybrid system for collision detection comprising the aforementioned

2 Related work

2.1 Collision detection

Collision-detection systems check a set of n objects for collision. Most are multiphase, but there are many ways to delineate these phases. This article will utilise the following two definitions:

> **Broad phase** This uses an approximate test to create a potentially colliding set comprising pairs of objects.
>
> **Narrow phase** This checks the potentially colliding set using a more accurate algorithm, and it may also compute the distance between objects as well as the point and time of collision.

Multiphase collision detection is based on the hypothesis that the broad phase's approximate test will eliminate the vast majority of objects from consideration. This scheme typically leads to a significant improvement in throughput.

The broad phase is concerned with bounding volumes. These are convex shapes that simplify complex and non-convex environment geometry. A plurality of bounding volumes exist. Spheres [1] are the same as their geometric counterparts, and their advantage is that they are invariant under rotation. Axis-aligned bounding boxes (AABBs) [2, 3] are cuboids whose axes are aligned with those of the environment. Oriented bounding boxes (OBBs) [4] extend AABBs by removing the axis-aligned requirement. Discrete-oriented polytopes (k-DOPs) [5] are k-sided parallelepipeds where the surfaces consist of hyperplanes whose normals belong to a fixed set of k vectors. There exist a number of algorithms to check these bounding volumes for collision. All-pairs checks every object for collision against every other object, resulting in $\frac{n(n-1)}{2}$ comparisons. An alternative is full-sort sweep and prune [2], which sorts axes to determine when a collision begins and ends. Incremental sweep and prune [3] improves on this by using insertion sort to exploit coherence. Spatial partitioning [6] is another alternative that uses grids to divide the environment into cells before placing each object within an appropriate cell. It reduces the number of pairwise collision tests by only checking objects within the same cell.

The narrow phase typically uses bounding-volume hierarchies (BVHs) [7]. BVH algorithms traverse these hierarchies to prune branches where a collision is impossible. Deformable narrow phases [8] attempt to refit BVHs to objects undergoing deformation. Recent research has attempted to improve the accuracy of these algorithms [9]. Continuous collision detection [10] is also a topic of current interest. These algorithms attempt to fit BVHs to the motion of objects so that collisions are not missed within the intervals between cycles. Alternatives to BVH traversal such as Lin-Canny [11] and V-Clip [12] work by tracking the closest features of polyhedra.

There has also been research interest in performing collision detection on GPUs. Originally, this research repurposed rasterisation to find overlapping objects [13]. As GPUs developed fully programmable cores, researchers moved to utilise these. Liu et al. [14] outline a broad phase that represents objects as a collection of spheres processed using spatial partitioning, followed by full-sort sweep and prune along a single axis chosen to minimise the number

of overlaps. The narrow phase is avoided using a penalty algorithm for rigid-body dynamics, although this can introduce substantial divergences from expected results. Combining this with the use of sweep and prune along only a single axis is likely to compound these inaccuracies. Significant accelerations are demonstrated, but the throughput starts to deteriorate above 128,000 objects, leading to scalability concerns. Avril, Gouranton and Arnaldi [15] outline an alternative broad phase that uses a hybrid system comprising CPU-based spatial partitioning and GPU-based full-sort sweep and prune. A novel mapping function and square root approximation logic avoid global memory accesses and reduce atomic operations. The authors demonstrate significant accelerations, but the rate declines as the object count increases, leading to the same scalability concerns as for the previous research. A narrow-phase algorithm is proposed by Lauterbach, Mo and Manocha [16]. This algorithm exploits GPU cores using a parallelised front-based traversal method. This method can be specialised for deformable objects [17]. A derivative exploiting GPU texture memory has been proposed by Zhang and Kim [18]. HPCCD [19] increases the level of parallelism by splitting a narrow-phase algorithm across a hybrid system comprising a CPU and GPU operating simultaneously. The CPU performs BVH traversal while the GPU executes elementary collision tests.

2.2 Ray tracing

We restrict our review of ray tracing to spatial-partitioning hierarchies, as these are the only element germane to this article. Traditionally, the most common hierarchy was the *k*-d tree [20]. *k*-d trees divide the environment by splitting along an arbitrary plane aligned to the world axes. There has recently been significant interest in BVHs for ray tracing [21, 22], which are not the same generalisable hierarchies used in collision detection but are instead AABB hierarchies constructed in accordance with the surface-area heuristic (SAH) metric. Their advantage is that, unlike *k*-d trees, they can be refitted in dynamic scenarios.

2.3 Microarchitectures

There is currently significant research interest in microarchitectures, as multicore architectures are expected to soon encounter a utilisation wall when they reach silicon power-density limits. This wall will limit the fraction of a processor that can run at full speed. It results from increasing transistor counts combined with an inability to reduce the power to switch a transistor. Esmaeilzadeh et al. [23] posit that at 8 nm, over 50 % of a processor will be unutilised. This will result in a 14 % throughput increase per annum, which is substantially less than current trends. The solution proposed by most researchers is specialisation [24–26], which involves offloading parallelisable computations to embedded microarchitectures. It has already been used effectively in a variety of processors. An increasing number of CPUs include specialised primitives [27], and the Apple iPhone 6 A8 comprises approximately 64 % fixed-function logic [28].

An early attempt at a collision-detection microarchitecture is outlined by Atay, Lockwood and Bayazit [29]. This exclusively focuses on the narrow phase and is designed for robotics. The triangle-triangle intersection test employed by the microarchitecture allows it to achieve high accuracy at the expense of interactivity. The CollisionChip [30] is an alternative narrow-phase microarchitecture that uses 24-DOP hierarchies storing triangles in the leaf nodes. It traverses a single hierarchy combining those of the two objects being tested, using an algorithm designed to reduce memory accesses and node transformations. A specialised separating-axis test (SAT) is used to test the *k*-DOPs and triangles for collision. The design is specialised for CAD objects with extremely large quantities of triangles and, like the previous microarchitecture, is not focused on achieving real-time results. An alternative approach is employed by the now discontinued AGEIA PhysX, which is a commercial IC and associated driver designed to accelerate physics including collision detection. A patent [31] outlines two possible designs, which both revolve around a specialised very-long instruction word (VLIW) processor with a plurality of floating-point units. It is ambiguous as to whether this system performs collision detection on the PhysX IC, the CPU or a combination of both.

2.4 Previous research

An earlier revision of our design [32] achieved an acceleration of 1.5×, despite being limited to 512 objects due to object duplication in memory. The current article builds on this by providing results for up to 1,024,000 objects, achieved via the integration of the microarchitectures into a complete hybrid system comprising the reuse of ray-tracing spatial-partitioning hierarchies. This article additionally compares and contrasts two alternative microarchitecture designs, and it provides the expected throughput if the microarchitectures were implemented on an ASIC.

3 Design

The design of our hybrid collision-detection system comprises three stages:

Spatial-partitioning broad phase This executes on a processor and divides the environment into cells.
Cell-based broad phase This utilises one of the two microarchitectures to perform collision detection on the contents of each cell.

Narrow phase This stage executes on a processor and performs conventional narrow-phase processing.

We begin by outlining the core element of our system, the cell-based broad phase, before complementing this with a description of the spatial-partitioning broad phase.

3.1 Cell-based broad phase

We chose a microarchitecture as the platform for our cell-based broad phase for three primary reasons.

Degree of parallelism Microarchitectures can accelerate algorithms with a high degree of parallelism. The broad phase offers significant scope for parallelisation with a workload that can be statically balanced to ensure consistently high utilisation throughout the microarchitecture.

Compute and memory bound Microarchitectures exploit parallelism to accelerate compute-bound algorithms but offer fewer advantages to memory-bound algorithms. The broad phase involves a high degree of computation with memory accesses that can be aggregated to reduce their impact.

Sequence of operations Consistent sequences of operations require minimal control logic and facilitate efficient pipelining through the reuse of standardised computation engines, thereby lending themselves to microarchitecture implementation. The broad phase tends to use standard, recurring collision tests performed in a consistent sequence.

The selection of a microarchitecture was also influenced by evidence that the broad phase consumes a considerable portion of the interactive application program loop. Lin and Gottschalk [33] and Fan et al. [34] discovered that collision detection is often a bottleneck. We calculated from the 22 benchmarks of the third experiment in Woulfe and Manzke [35] that a mean of 47 % of the overall collision-detection time is spent in the broad phase. This calculation can be considered a conservative estimate, as only the high throughput dynamic bounding-volume tree (DBVT) algorithm from the Bullet Physics SDK was considered. Finally, it should be noted that even if the broad phase were not to account for a major part of the program loop in a given scenario, the microarchitecture would still provide throughput improvements that would facilitate increased realism.

To design the microarchitectures, we began by investigating the various broad-phase algorithms. The most commonly used is incremental sweep and prune. However, it is difficult to parallelise across more threads of execution than the number of coordinate axes. One solution would be to switch from incremental to full-sort sweep and prune, but this essentially obviates the algorithm's primary advantage of coherence. The GPU sweep-and-prune implementations designed by Liu et al. [14] and Avril, Gouranton and Arnaldi [15] represent an attempt to trade coherence for parallelism. Despite the promise shown, full-sort sweep and prune would not be entirely amenable to microarchitectures as it makes significant use of sorting. Sorting tends to be problematic due to the overhead of memory access latencies and, therefore, tends to either inadequately exploit parallelism [36] or have undesirable throughput-to-area trade-offs [37]. Harkins et al. [38] claim that algorithms utilising sorting are not suited to microarchitecture implementation. For these reasons, sweep and prune would be a suboptimal choice. This corresponds to the findings of Chen et al. [39] that the best serial algorithms can have poor parallel scalability.

In contrast, we discovered that all-pairs is ideal for microarchitecture implementation. It is embarrassingly parallel, and this parallelism can be used to effectively exploit resources. AABBs were selected as the bounding volumes, since they tend to provide a good object fit while requiring a relatively low quantity of arithmetic components, thereby enabling many operations to be performed in parallel. AABBs have also been successfully used by the I-COLLIDE [3] and SOLID [7] libraries. A sequential version of the algorithm is:

function ALLPAIRSAABB
  n: Object count
  $\min_a^b$: Minimum of AABB a along axis b
  $\max_a^b$: Maximum of AABB a along axis b
  for $i \leftarrow 1$ **to** $n-1$ **do**
    for $j \leftarrow i+1$ **to** n **do**
      collision $\leftarrow$
        $\left(\max_i^x \geq \min_j^x\right) \wedge \left(\min_i^x \leq \max_j^x\right)$
        $\wedge \left(\max_i^y \geq \min_j^y\right) \wedge \left(\min_i^y \leq \max_j^y\right)$
        $\wedge \left(\max_i^z \geq \min_j^z\right) \wedge \left(\min_i^z \leq \max_j^z\right)$
      if *collision* **then**
        result $\leftarrow$ *result* $\cup \langle i, j \rangle$
      end if
    end for
  end for
  return *result*
end function

Another advantage of all-pairs is that its throughput is deterministic. In contrast, sweep and prune has a computational complexity of $O(n+s)$, where s denotes the number of swapping operations required to maintain the algorithm's sorted object lists. As s cannot be determined a priori, the behaviour of sweep and prune can vary significantly. Scenarios with many moving objects

result in significant increases in s that lead to decreases in throughput. Tracy, Buss and Woods [40] demonstrate that scenarios with few moving objects can also perform poorly if the total number of objects is very high. Only all-pairs facilitates the accurate deduction of the most complex scenario that can be executed within a given timeframe, without concern that the frame rate will decrease in certain scenarios. All-pairs unlocks the possibility of a wider range of scenarios through the avoidance of the non-deterministic throughput of sweep and prune.

3.1.1 Area-efficient microarchitecture

The first design of the cell-based broad-phase microarchitecture is area efficient. In other words, it uses a minimal quantity of resources to exploit available parallelism. However, the trade-off is that it is limited in the quantity of objects supported. The design consists of a pipeline implementing two primary operations—buffer and compare. A schematic is provided in Fig. 1.

As the availability of resources can vary, the microarchitecture is designed to be extensible via a factor m. When many resources are available, the design can take full advantage of these to gain the maximum achievable acceleration, while it will still fit and execute efficiently when resources are constrained.

The microarchitecture could represent numbers using fixed-point formats, but these have relatively low accuracy. Moreover, their economical use of resources would offer little advantage as the limiting factor tends to be the quantity of memory and not the quantity of logic consumed. In addition, effective use of pipelining almost entirely eliminates the throughput gains that could be achieved. Therefore, the microarchitecture represents numbers using the single-precision IEEE 754 floating-point format. There is a wealth of research highlighting the efficiency of performing floating-point computations on microarchitectures [41]. The number of mainstream libraries defaulting to single precision, such as SOLID [7] and Bullet, indicates that this offers sufficient accuracy. All platforms can use this format, precluding the need to translate when communicating data.

Buffer The buffer stores each AABB's data in an efficient manner for processing by the subsequent compare operation. During initialisation, the buffer reads each AABB and stores the data in $6m$ internal dual-port memories. The $6m$ memories correspond to m memories for each of the minimum x, maximum x, minimum y, maximum y, minimum z and maximum z values. The data are replicated across each set of m memories, so that each of the m memories contains the same data. This results in six logical $2m$-port memories, allowing $12m$ data to be outputted in a single clock cycle.

In the following sections, the first port of each dual-port memory will be referred to as A and the second port will be referred to as B. The six memories that contain each AABB's data and that share an index will be referred to as a *memory group*. For example, memory group 0 contains minimum x memory 0, maximum x memory 0, minimum y memory 0, maximum y memory 0, minimum z memory 0 and maximum z memory 0. In the following sequence, the inputs to each memory belonging to a given memory group remain the same at all times.

To enable the required sequence of object-object comparisons, the AABBs are outputted from the memory

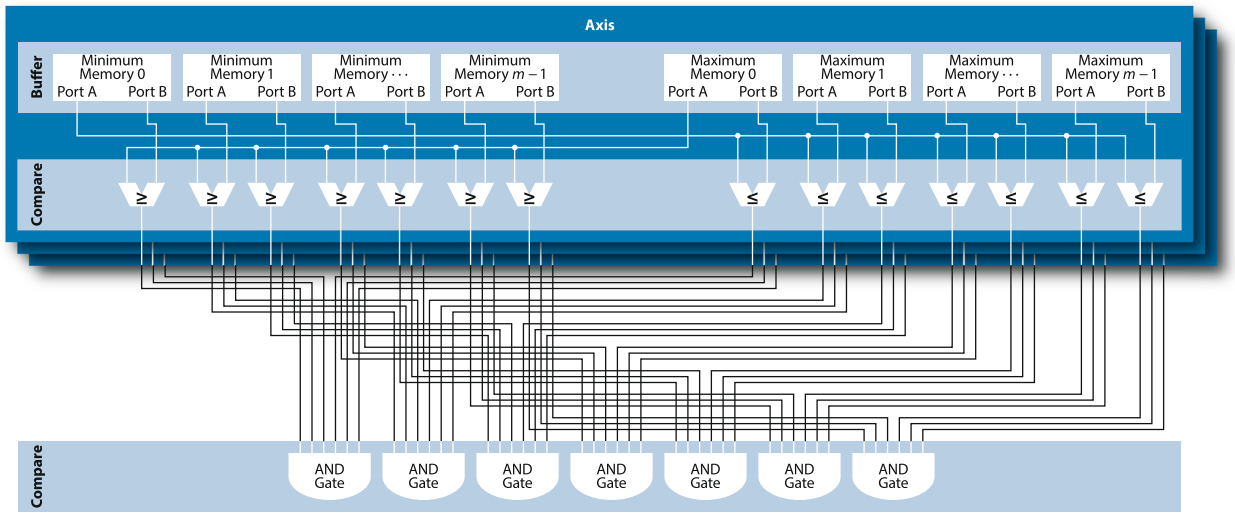

Fig. 1 Schematic of the area-efficient microarchitecture. The buffer and the comparators from the compare operation are replicated three times to cover the three axes

groups in a specific sequence. Initially, the address input to memory group 0's A is set to 0, while the remaining $2m-1$ are set to the subsequent addresses. On the subsequent cycle, 0's A retains its value, and the remaining are each incremented by $2m-1$. This sequence continues until any input selects $n-1$. At this stage, 0's A is set to 1, while the remaining are set to the subsequent addresses. The sequence continues until 0's A selects $n-2$. Addresses after $n-1$ may be accessed using this sequence; these must be subsequently removed. The sequence is exemplified in Table 1.

In this proposal, the parallelism per cycle varies. It would be preferable to maintain a consistent high level of parallelism throughout execution, and a variety of schemes could be used to achieve this. However, although practical in theory, these schemes become impossible to implement in a microarchitecture, as the complexity of the required control logic would consume large quantities of resources and the design would fail to achieve an adequate clock frequency. Through experimentation, we selected the outlined design as the variable parallelism is compensated for by the ability to maintain a high clock frequency.

In the buffer, the memory bandwidth is $2fmw$ bit/s, where f is the clock frequency of the microarchitecture in hertz and w is the bit-width of a single memory location. The number of cycles required to generate the sequence is

$$\sum_{i=0}^{n-2} \left\lceil \frac{n-i-1}{2m-1} \right\rceil .$$

Compare The compare operation performs the comparison from all-pairs using the data supplied by the buffer. It compares the data outputted by memory group 0 with the data outputted by all other memory groups. It comprises $6m-3$ greater-than-or-equal-to and $6m-3$ less-than-or-equal-to comparators. The outputs are connected to $2m-1$ logical AND gates. Each gate takes six inputs corresponding to the six comparator results forming an AABB pair. If a collision is detected, the indices of the two colliding objects are written to a single line of memory.

3.1.2 Many-object microarchitecture

The second design of the cell-based broad-phase microarchitecture supports significantly greater object quantities. It achieves this advantage through the use of additional resources to avoid data replication. It, therefore, exhibits less parallelism. The design consists of a pipeline implementing three primary operations—buffer, reorder and compare. A schematic is provided in Fig. 2. This microarchitecture is also extensible via a factor m and also uses single-precision floating point.

Buffer During initialisation, the buffer works in the same way as for the area-efficient microarchitecture. It reads each AABB and stores the data in $6m$ dual-port memories. However, unlike the area-efficient microarchitecture, these data are stored across the m memories in a format that precludes data duplication with, for example, the first AABB stored in the first address of the first memory group and the second AABB stored in the first address of the second memory group. This data layout, which is exemplified in Table 2, results in a schism between the memory group address and the index of the AABB being retrieved; the index can be computed using $am+j$ where a is the address and j is the memory group being accessed.

Table 1 Area-efficient microarchitecture sequence

	Address								Comparison						
Cycle	0A	0B	1A	1B	2A	2B	3A	3B	0–1	0–2	0–3	0–4	0–5	0–6	0–7
1	0	1	2	3	4	5	6	7	0–1	0–2	0–3	0–4	0–5	0–6	0–7
2	0	8	9						0–8	0–9					
3	1	2	3	4	5	6	7	8	1–2	1–3	1–4	1–5	1–6	1–7	1–8
4	1	9							1–9						
5	2	3	4	5	6	7	8	9	2–3	2–4	2–5	2–6	2–7	2–8	2–9
6	3	4	5	6	7	8	9		3–4	3–5	3–6	3–7	3–8	3–9	
7	4	5	6	7	8	9			4–5	4–6	4–7	4–8	4–9		
8	5	6	7	8	9				5–6	5–7	5–8	5–9			
9	6	7	8	9					6–7	6–8	6–9				
10	7	8	9						7–8	7–9					
11	8	9							8–9						

An exemplar of the sequencing of the dataflow through the area-efficient microarchitecture with extensibility factor $m=4$ and object count $n=10$. On each clock cycle, the microarchitecture requests the specified memory addresses from the memory groups and ports indicated. These data are subsequently compared according to the comparison sequence outlined

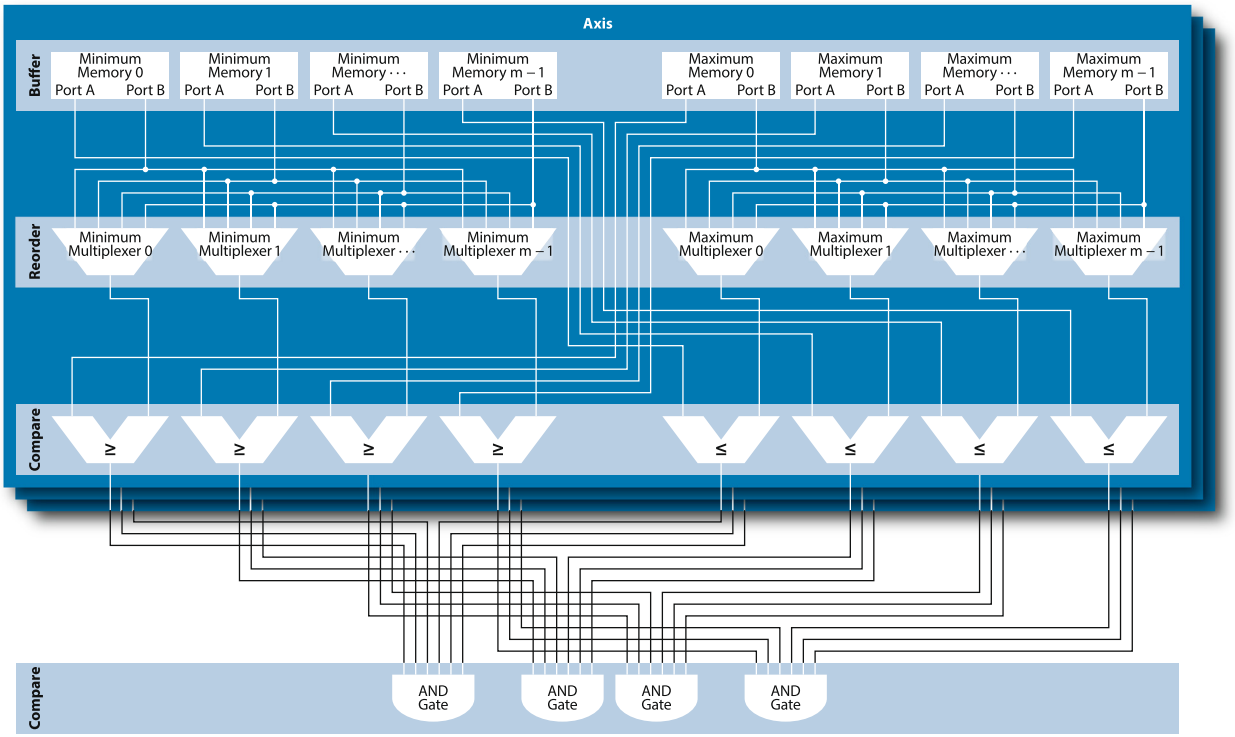

Fig. 2 Schematic of the many-object microarchitecture. The buffer, the reorder and the comparators from the compare operation are replicated three times to cover the three axes

As for the area-efficient microarchitecture, the AABBs are outputted from the memory groups in a specific sequence. Initially, all memory groups' A address inputs are set to 0 while 0's B is set to 1. The remaining Bs are set to 0. On the subsequent cycle, 1's B is incremented to 1, and the other Bs retain their previous values. This sequence continues until 0's B selects $\lceil \frac{n}{m} \rceil$, which ensures that all comparisons to AABB $n-1$ have been performed. At this stage, all As are set to 1, 0's B is set to 2, and the remaining Bs are set to 1. The sequence continues until some A selects $\lceil \frac{n}{m} \rceil - 1$, which is the address corresponding to AABB $n-2$, and 0's B selects $\lceil \frac{n}{m} \rceil$. The sequence is exemplified in Table 3. As for the area-efficient microarchitecture, this design exhibits a variable degree of parallelism, as addresses after $n-1$ may be accessed; the outlined solution represents a compromise between parallelism and clock frequency.

Table 2 Many-object microarchitecture buffer layout

	Object index			
Address	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9		

An exemplar indicating the layout of the objects in the buffer operation of the many-object microarchitecture with extensibility factor $m = 4$ and object count $n = 10$

In the buffer, the memory bandwidth is $2fmw$ bit/s. The number of cycles required to generate the sequence is

$$\sum_{i=0}^{\lceil \frac{n}{m} \rceil - 1} (n - im - 1)\,.$$

Reorder If the data emitted from the buffer were immediately sent to the compare operation, only a fraction of the required comparisons would take place and some of these would be repeated. The goal of the reorder operation is to rectify this using $6m$ multiplexers to create the appropriate sequence of comparisons. These multiplexers consume significant resources, which is the reason this microarchitecture is less area efficient than the previous.

Following from the definition of a memory group, we use the term *multiplexer group* to denote a set of six multiplexers sharing the same index. For example, multiplexer group 0 comprises minimum x multiplexer 0, maximum x multiplexer 0, minimum y multiplexer 0, maximum y multiplexer 0, minimum z multiplexer 0 and maximum z multiplexer 0.

On initialisation of the reorder operation, multiplexer group 0's selector is set to 1, 1's is set to 2, $m-2$'s is set to $m-1$ and $m-1$'s is set to 0. On each cycle, every selector

Table 3 Many-object microarchitecture sequence

	Address				Object index				Address				Object index				Multiplexer			
Cycle	0A	1A	2A	3A	0A	1A	2A	3A	0B	1B	2B	3B	0B	1B	2B	3B	0	1	2	3
1	0	0	0	0	0∗	1†	2‡	3§	1	0	0	0	4§	1∗	2†	3‡	1	2	3	0
2	0	0	0	0	0∗	1†	2‡	3§	1	1	0	0	4‡	5§	2∗	3†	2	3	0	1
3	0	0	0	0	0∗	1†	2‡	3§	1	1	1	0	4†	5‡	6§	3∗	3	0	1	2
4	0	0	0	0	0∗	1†	2‡	3§	1	1	1	1	4∗	5†	6‡	7§	0	1	2	3
5	0	0	0	0	0∗	1†	2‡	3§	2	1	1	1	8§	5∗	6†	7‡	1	2	3	0
6	0	0	0	0	0∗	1†	2‡	3§	2	2	1	1	8‡	9§	6∗	7†	2	3	0	1
7	0	0	0	0	0∗	1†	2‡	3§	2	2	2	1	8†	9‡	§	7∗	3	0	1	2
8	0	0	0	0	0∗	1†	2‡	3§	2	2	2	2	8∗	9†	‡	§	0	1	2	3
9	0	0	0	0	0∗	1†	2‡	3§	3	2	2	2	§	9∗	†	‡	1	2	3	0
10	1	1	1	1	4∗	5†	6‡	7§	2	1	1	1	8§	5∗	6†	7‡	1	2	3	0
11	1	1	1	1	4∗	5†	6‡	7§	2	2	1	1	8‡	9§	6∗	7†	2	3	0	1
12	1	1	1	1	4∗	5†	6‡	7§	2	2	2	1	8†	9‡	§	7∗	3	0	1	2
13	1	1	1	1	4∗	5†	6‡	7§	2	2	2	2	8∗	9†	‡	§	0	1	2	3
14	1	1	1	1	4∗	5†	6‡	7§	3	2	2	2	§	9∗	†	‡	1	2	3	0
15	2	2	2	2	8∗	†	‡	§	3	2	2	2	§	9∗	†	‡	1	2	3	0

An exemplar of the sequencing of the dataflow through the many-object microarchitecture with extensibility factor $m = 4$ and object count $n = 10$. On each clock cycle, the microarchitecture requests the specified memory addresses from the memory groups and ports indicated. These memory addresses result in the outputting of the specified object indices. The symbols ∗, †, ‡ and § indicate the indices used in each comparison performed within the microarchtecture's compare operation, which are chosen using the multiplexer selectors specified

is incremented by 1 mod m. The sequence restarts any time the buffer's A is modified. This is exemplified in Table 3.

Compare The compare operation comprises $3m$ greater-than-or-equal-to and $3m$ less-than-or-equal-to comparators. The outputs are connected to m logical AND gates. Each gate takes six inputs corresponding to the six comparator results forming an AABB pair. If a collision is detected, the indices of the two colliding objects are written to a single line of memory.

3.2 Spatial partitioning

Despite the microarchitectures' effective exploitation of parallelism and bandwidth, there are two potential concerns. The first concern is that the depth of the memories results in a restriction on the quantity of bounding volumes and, therefore, on the quantity of objects. Although many microarchitecture memories are now of substantial depth, the imposition of any such limit could be considered unsatisfactory. The second concern is that all-pairs suffers from an undesirable computational complexity of $\mathrm{O}\left(n^2\right)$, resulting from the algorithm's non-exploitation of coherence. Neither issue affects scenarios of small or moderate size, and the microarchitectures operating in isolation are sufficient to accelerate these. However, it is desirable to find a solution to these issues in order to unlock the possibility of larger scenarios.

Our solution is to transform the broad phase into a hybrid system combining the microarchitectures with a processor. This processor executes spatial partitioning to divide the list of objects into appropriately sized cells for microarchitecture processing. It has the primary advantages of overcoming object limits and reducing computational complexity. An auxiliary advantage is the possibility of increased parallelism through the overlapping of computations performed by the different stages. Once the potentially colliding set corresponding to a cell is received from the microarchitecture, narrow-phase processing of the cell can proceed while the microarchitecture processes the subsequent cell.

However, this new stage could consume additional computational resources and negatively affect overall system throughput. To ameliorate this issue, we reuse the hierarchies from ray tracing. Reusing an existing data structure offers a significant reduction in construction costs and memory footprint. Moreover, by selecting ray-tracing hierarchies, we benefit from the current high degree of research interest in ray tracing, while aligning with the direction in which graphics applications are ultimately heading.

One significant difference exists in the way ray tracing optimally consumes hierarchies and the way our microarchitectures optimally consume them. For ray tracing's broad phase, it is usually beneficial to subdivide hierarchy branches as far as possible, aiming

to achieve approximately one object per leaf. The ray tracer will use the generated leaf nodes to perform ray-object culling. This high degree of subdivision is not beneficial for the microarchitectures; they are designed to efficiently process moderate quantities of objects to effectively exploit parallelism. To reconcile this difference, we retain, during construction of the hierarchy, the leaf nodes with a quantity of objects less than or equal to the selected microarchitecture object limit. Once broad-phase collision-detection processing commences, the contents of the recorded cells are accessed by the microarchitectures.

Most contemporary ray tracers use BVHs, but these are not entirely suitable for collision detection as they permit objects to overlap cell boundaries. These overlapping objects would need to be resolved before performing collision detection, and this resolution would nullify many of the benefits of reuse. One ray-tracing hierarchy that provides a solution is the *k*-d tree, as this hierarchy places objects that overlap cell boundaries within all overlapped cells. Although a *k*-d tree would offer excellent throughput for collision detection, it would be less desirable for ray tracing, as *k*-d trees cannot be easily refitted for dynamic scenarios. To reconcile the throughput of BVHs with the flexibility of *k*-d trees, we propose a two-level hierarchy. Our proposal consists of a *k*-d tree that is subdivided until each leaf node contains a quantity of objects less than or equal to the microarchitecture object limit. Within each leaf, a BVH splits the cells until each contains a single object in accordance with ray-tracing practice. The two-level hierarchy is not time consuming to construct, as only relatively few levels of the *k*-d tree are required, and the throughput degradation is negligible as they can be constructed in $\mathrm{O}\left(n \log n\right)$ [20]. In some cycles of the interactive application program loop, it may be necessary to perform slight alterations to the *k*-d tree if the position of objects changes significantly. This could require migration of objects between cells, but the large cell sizes mean migration will occur infrequently and the cost will be negligible. It is, furthermore, unlikely that the quantity of objects in a cell will precisely equal the cell-size limit, thereby allowing cells to accommodate additional objects without rebuilding or refitting in many cases. The underlying BVHs serve to maximise throughput as they can be efficiently refitted on each cycle. Therefore, the proposed two-level hierarchy maximises the throughput of both collision detection and ray tracing.

4 Implementation

We envisage our cell-based broad-phase microarchitectures fabricated as part of an IC that would also execute the remainder of the interactive application program loop. Using a single platform would allow for the elimination of data transfer overheads. This concept has been successfully adopted to integrate a CPU and GPU within some Intel Core processors [42] as well as AMD accelerated processing units (APUs) [43], such as those in the PlayStation 4. Within the spectrum of platforms readily available today, our microarchitectures could naturally reside within the fixed-function logic of GPUs, as there is already a significant focus on relocating many elements of the interactive application program loop to these platforms [44]. The remainder of the program loop could utilise the programmable elements of the GPU. Adding one of the microarchitectures would not compromise GPU programmability, as all GPUs include some fixed-function logic such as rasterisation. This is unlikely to change due to power-density limits as well as the lacklustre throughput achieved when traditionally fixed-function elements have been reimplemented using the programmable elements of a GPU [45]. Moreover, it is not prohibitively expensive to include one of the microarchitectures, as the large production volumes of commodity platforms amortises the cost [27]. Therefore, there exists sufficient motivation for the fabrication of our logic as part of a future GPU.

These platforms were unavailable to us, and we were limited to prototyping our microarchitectures on a field-programmable gate array (FPGA), to which we translated, mapped, placed and routed a complete design written in hardware-description language (HDL). FPGAs are ICs that are reconfigurable, meaning that a single FPGA can implement different microarchitectures at different times. However, this reconfigurability incurs significant throughput, power and area penalties.

One of the primary characteristics of the microarchitectures is the possibility of their adaptation to the size of the underlying platform. We found that the limiting factor for both microarchitectures was the quantity of internal memories, which constrained both designs to $m = 16$. Based on this value, it was possible to process a maximum of 1024 objects using the area-efficient microarchitecture and a maximum of 16,384 objects using the many-object microarchitecture.

When targeting platforms such as GPUs, the FPGA can be used to verify the functionality of the microarchitectures and to analyse their behaviour, but it is insufficient for gaining a true reflection of throughput. To address this, we adapted the throughput metrics from the FPGA implementations to a clock frequency of 500 MHz in accordance with assumptions made in existing research [46–48]. Since 500 MHz is significantly lower than the clock frequencies of modern GPUs, we, therefore, derive a conservative estimate of throughput. We excluded the communication overhead as we intend that our microarchitectures would reside on the same IC as all associated computation. All other elements retained the same values as their FPGA counterparts. In practice, however, it

is likely that the constraints on internal memory sizes would be less severe, thereby facilitating the possibility of simulating many more objects without spatial partitioning. In addition, it is likely that m could be increased due to the greater density of resources. Therefore, all throughput metrics are conservative estimates of what would be achieved in a practical system centred around a GPU.

Our CPU-based software utilised the Bullet Physics SDK. We added custom C++ code to adapt the broad phase to gather the relevant data from the ray-tracing hierarchies, before invoking the appropriate microarchitecture operations and reading the resultant potentially colliding sets.

5 Results and discussion

The adapted Bullet code was compiled using G++ with throughput optimisations enabled. The host system consisted of a Quad-Core AMD Opteron 2350 clocked at 2 GHz with 8 GB of RAM. The operating system was 64-bit Ubuntu Linux. Our GPU results were measured using an NVIDIA GeForce GTX 670 with 2 GB of external memory.

Our experiments used an updated version of the framework for benchmarking collision detection [35]. Our benchmarks consisted of 1000 collision-detection cycles of a scenario comprising a cube enclosing n objects, as illustrated in Fig. 3. The dimensions of the cube were $\sqrt[3]{50^3 \times 5n}$ m. The objects were uniformly distributed throughout the environment, and all object properties were determined using the uniform probability distribution with different values possible for each axis. The objects were spheres, cuboids, cylinders and cones, and their sizes lay between 25 and 75 m. The linear velocity spanned from $(-25, -25, -25)$ to $(25, 25, 25)$ m/s, and the angular velocity spanned from $(-2.5, -2.5, -2.5)$ to $(2.5, 2.5, 2.5)$ rad/s. In all of the benchmarks, our goal was to generate a large quantity of objects undergoing

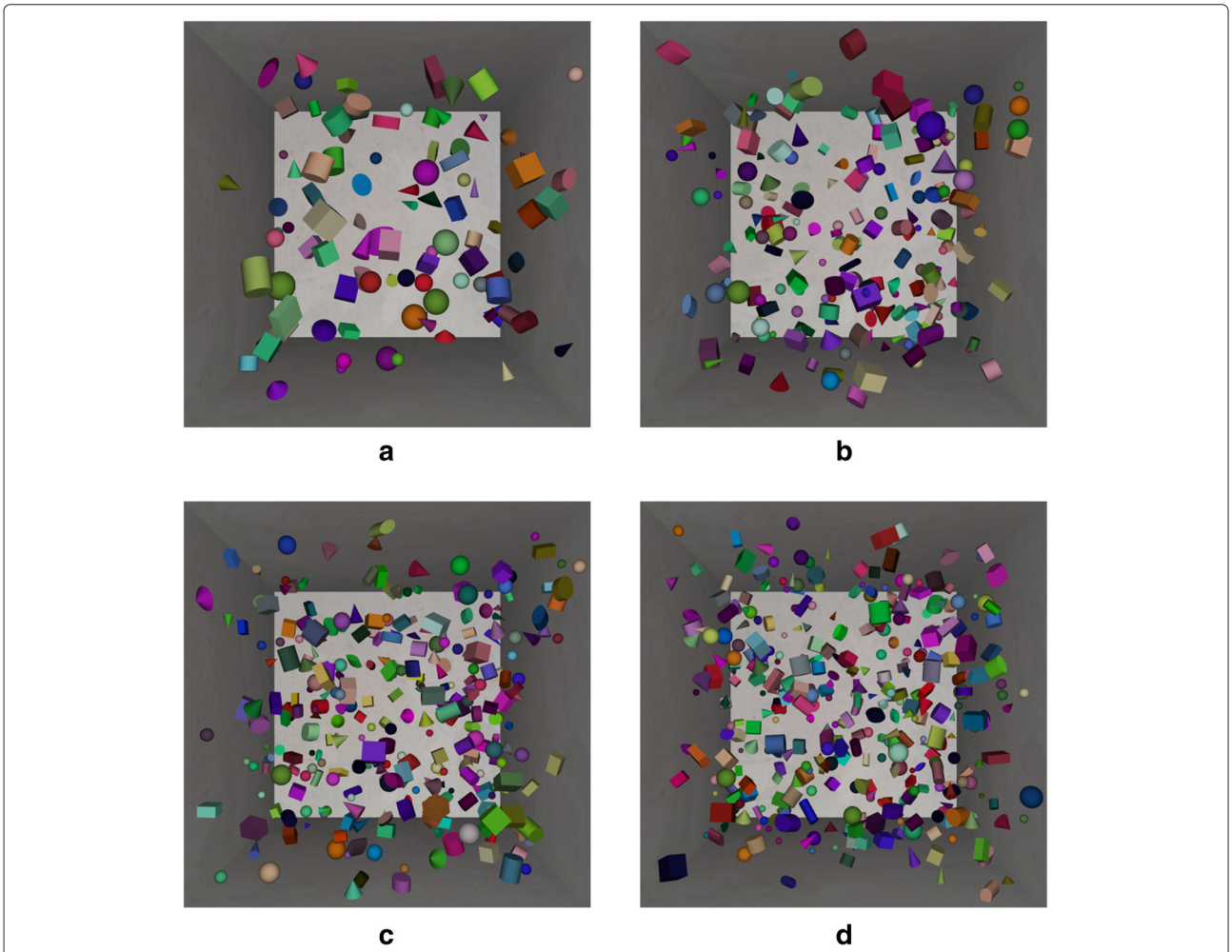

Fig. 3 Sample benchmarks. **a** 100 objects. **b** 200 objects. **c** 300 objects. **d** 400 objects

a high degree of movement and interaction, in order to thoroughly assess throughput under what are traditionally considered the most challenging cases for broad-phase algorithms. In addition, we used relatively simple objects, as the broad phase is only concerned with objects and ignores details such as their complexity.

The throughput was computed using 20 variants of the benchmark. For our system, only the microarchitecture computations were recorded; the spatial-partitioning hierarchy construction was excluded as this would form part of a ray tracer and would not add to the system's execution time. A variety of cell sizes—1024 for the area-efficient microarchitecture and 1024, 2048, 4096, 8192 and 16,384 for the many-object microarchitecture—was tested to determine the optimal. Throughput counters embedded in the microarchitectures were used to record the timing metrics. For comparison purposes, we measured the execution times of broad-phase algorithms executing on a CPU and GPU. For the CPU comparison, we selected Bullet's DBVT algorithm, for which we recorded the execution times using high-resolution throughput counters present in the CPU. We selected this algorithm as it achieves optimal throughput for the vast majority of scenarios. It performs spatial partitioning using two AABB hierarchies whose nodes can be dynamically rearranged. One hierarchy represents static objects and the other represents dynamic objects. The objects are moved between the two hierarchies as their statuses change. For the GPU comparison, we selected Bullet's GPU-based sweep and prune, for which we recorded the time spent checking for collisions between objects, while excluding the transfer of data between the CPU and GPU. It is an implementation of the design proposed by Liu et al. [14]. The execution times of all these systems are tabulated in Table 4, and the acceleration factors of our system relative to the CPU and GPU are plotted in Fig. 4.

All of our results indicate that very significant accelerations can be achieved, as all benchmarks have been significantly accelerated relative to both the CPU and GPU. The results also show that the area-efficient microarchitecture outperforms the many-object microarchitecture by a factor of 3.14×. The higher degree of acceleration can be explained by the greater quantity of comparisons performed in parallel. Using the results from the area-efficient microarchitecture, it can be observed that for those benchmarks involving no spatial partitioning, the

Table 4 Execution times for the CPU, GPU, area-efficient microarchitecture and many-object microarchitecture

			Area-efficient microarchitecture		Many-object microarchitecture					
Objects	CPU	GPU	None	1024	None	1024	2048	4096	8192	16,384
100	0.2213	0.9732	**0.0005**		**0.0008**					
200	0.4478	0.9982	**0.0016**		**0.0029**					
300	0.7990	1.0347	**0.0033**		**0.0062**					
400	0.9994	1.0441	**0.0057**		**0.0107**					
500	1.2966	1.0302	**0.0087**		**0.0165**					
600	1.5770	1.0727	**0.0124**		**0.0236**					
700	1.9871	1.0796	**0.0168**		**0.0319**					
800	2.5760	1.0889	**0.0217**		**0.0414**					
900	2.9700	1.1072	**0.0273**		**0.0522**					
1000	3.1694	1.0945	**0.0336**		**0.0643**					
2000	6.0023	1.2040		**0.0423**	0.2535	**0.0804**				
4000	14.2332	1.4039		**0.0913**	1.0070	**0.1727**	0.4036			
8000	30.7177	1.9585		**0.1665**	4.0140	**0.3153**	0.6270	1.7638		
16,000	69.8204	3.6801		**0.3074**	16.0280	**0.5821**	1.2199	2.5936	8.2487	
32,000	159.5770	8.3066		**0.5676**		**1.0749**	2.2525	4.7891	15.2312	36.9132
64,000	375.3380	22.8188		**1.0481**		**1.9848**	4.1592	8.8431	28.1244	68.1603
128,000	834.9720	67.5312		**1.9353**		**3.6649**	7.6800	16.3288	51.9317	125.8579
256,000	1940.2600	187.3310		**3.5735**		**6.7672**	14.1811	30.1512	95.8919	232.3967
512,000	4423.3300	605.7710		**6.5985**		**12.4957**	26.1853	55.6742	177.0645	429.1206
1,024,000	9897.0100	1958.8776		**12.1841**		**23.0732**	48.3512	102.8025	326.9497	792.3714

All execution times are in milliseconds. 'CPU' is Bullet's DBVT, 'GPU' is Bullet's GPU sweep and prune, and the numeric table headers indicate the spatial-partitioning cell size used with the microarchitectures. Bold results highlight the optimal time for each microarchitecture

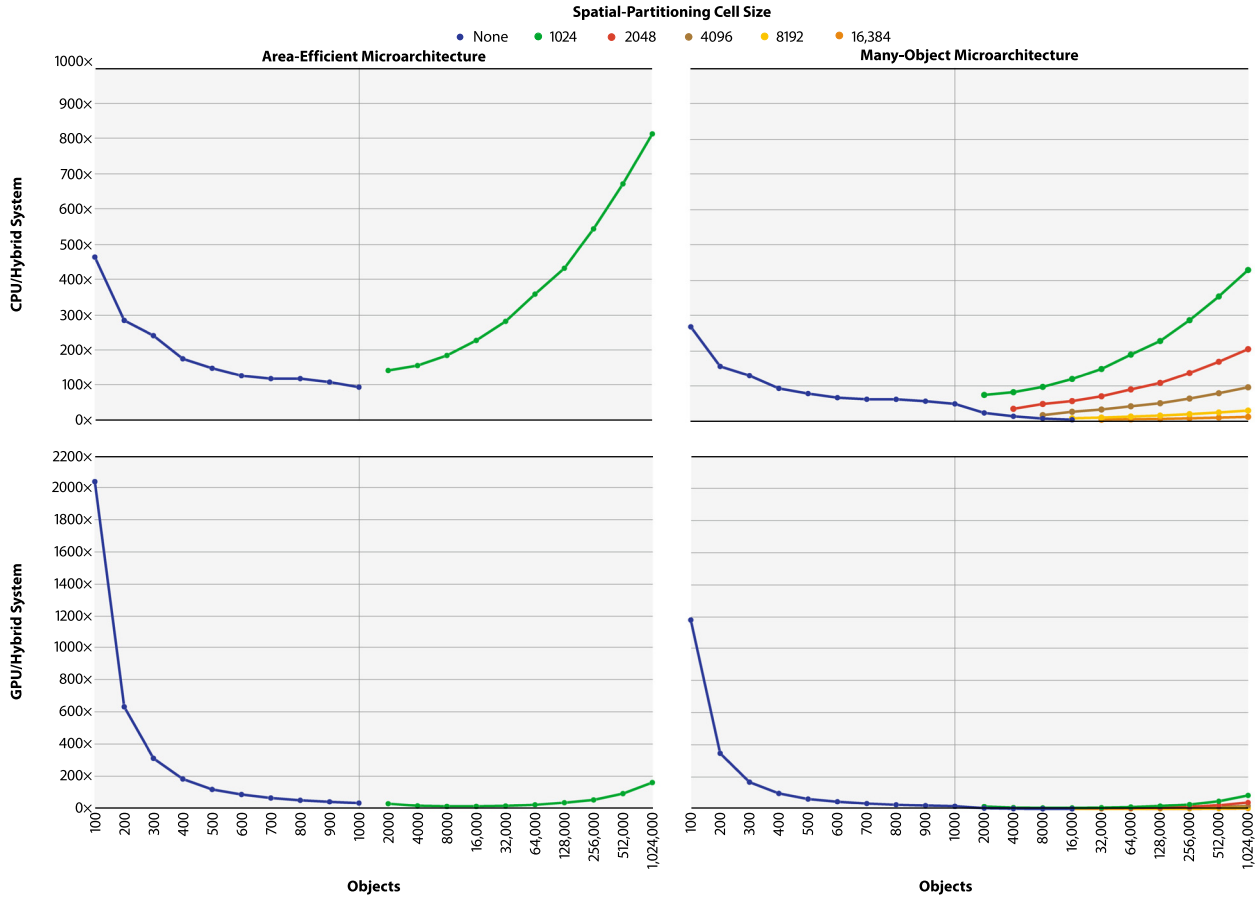

Fig. 4 Acceleration factors of the hybrid system relative to the CPU and GPU. The *vertical lines* indicate a change of scale along the *x* axis

acceleration never decreases below 94× relative to the CPU or below 33× relative to the GPU. If the results comprising spatial partitioning are included, the accelerations increase further. Compared to the CPU, our system achieves an acceleration of up to 812×, while compared to the GPU, it achieves an acceleration of up to 161×. In both cases, these accelerations correspond to the highest object counts. The results indicate that our system provides significant advantages to all benchmarks, both with and without spatial partitioning, and that the most challenging benchmarks for traditional collision detection are those most accelerated by our system.

The results also indicate that spatial partitioning has a beneficial effect on throughput. Without spatial partitioning, an $O(n^2)$ computational complexity can be observed. This complexity can be disregarded for up to 1000 objects, as it is minimal and has little impact due to the significant accelerations provided by parallelism. However, it is apparent that the complexity may affect throughput for larger object quantities. This is rectified by spatial partitioning, which our results indicate lowers the computational complexity to $O(n \log n)$. The positive effect is most evident when comparing against the CPU; it is less evident when comparing against the GPU as the GPU throughput is relatively poor for benchmarks with 1000 objects or fewer. The results also indicate that the optimal cell size is 1024, irrespective of microarchitecture. This is the optimal as the microarchitectures execute efficiently with moderate quantities of objects. Since both microarchitectures accommodate this cell size, we recommend selecting the area-efficient microarchitecture in all cases due to its enhanced throughput.

The results also demonstrate that our system is not significantly affected by object count, even when the count greatly exceeds what can be typically expected. This demonstrates that our designs do not need optimisation or specialisation for specific scenarios. In contrast, there are certain object-count ranges for which the CPU and GPU perform poorly. Higher object counts significantly degrade CPU throughput. The GPU tends to perform poorly with both large object counts which cannot be processed efficiently, as well as small object counts which inadequately exploit parallelism.

We further compare the previously provided throughput figures from the area-efficient microarchitecture with the throughput figures provided within two recent GPU broad-phase collision-detection articles in Table 5 and Fig. 5. The results demonstrate that our system significantly outperforms all of the algorithms for all object counts, with up to 14× acceleration over Liu et al. [14] and up to 40× acceleration over Avril, Gouranton and Arnaldi [15]. These results are not as representative of expected throughput as our prior results, since the benchmarks used in the articles differ significantly from those that we used. Our benchmarks included a high degree of object movement and interaction, to thoroughly assess the most challenging scenarios and prove that use of our system facilitates scenarios traditionally avoided due to their potentially negative impact on throughput. In contrast, many of the benchmarks used by the aforementioned articles choose scenarios that are more favourable to algorithms sensitive to object movement and interaction. Irrespective of these differences, the results clearly demonstrate that our system significantly outperforms the state of the art.

We previously mentioned that we envisage our microarchitectures implemented as fixed-function logic on a platform such as a GPU. To assess how the microarchitectures would fit, we computed the approximate size if implemented on the NVIDIA GeForce GTX TITAN X, using information on process widths, die sizes and transistor counts for our FPGA and the GPU, as well as research into the difference in area consumption between FPGA and ASIC implementations [49]. We deduced that the area-efficient microarchitecture would consume approximately 0.072 % of the GPU die area, while the many-object microarchitecture would consume approximately 0.086 %. It would also be useful to have assessed the expected power consumption of the microarchitectures when implemented on a GPU. Unfortunately, there were insufficient data available to accurately compute this information. However, a substantive body of research [23–26] indicates that specialised logic consumes significantly less power than programmable logic due to the removal of overheads such as instruction processing, register accesses and inefficient memory layouts. Analyses indicate a 16× power decrease can be expected, with up to a 500× decrease possible for some applications [50]. It is, therefore, evident that the microarchitectures would reduce power consumption appreciably.

Table 5 Execution time comparisons for recent GPU broad-phase collision-detection algorithms against the area-efficient microarchitecture

Article	Objects	Article execution times	Microarchitecture execution times
Liu et al.	16,000	3.0000	**0.3074**
	128,000	11.0000	**1.9353**
	960,000	161.0000	**11.4859**
Avril, Gouranton and Arnaldi	1000	1.3500	**0.0336**
	5000	1.3900	**0.1101**
	10,000	2.4100	**0.2017**
	50,000	10.2100	**0.8379**
	100,000	22.5200	**1.5471**
	500,000	116.7200	**7.5946**
	1,000,000	245.9500	**11.9223**

All execution times are in milliseconds. Bold results highlight the optimal time for each object count

6 Conclusions

We presented a hybrid system comprising one of two fixed-function microarchitectures complemented by a general-purpose microprocessor. The objective was to achieve significant acceleration and energy efficiencies for collision detection through the effective exploitation of parallelism and memory bandwidth. The area-efficient microarchitecture focused on occupying minimum resources, while the many-object microarchitecture focused on supporting greater object quantities. To compensate for potential challenges, such as a limit on the quantity of objects supported, the microarchitectures were combined with a spatial-partitioning phase executing on a processor. We outlined a novel means of combining this spatial-partitioning phase with the hierarchies constructed by ray tracers, thereby reducing computation time and memory consumption.

Overall, we demonstrated a significant enhancement in the throughput of collision detection, with a 1,024,000-object benchmark demonstrating an acceleration of 812× when compared against a CPU and an acceleration of 161× when compared against a GPU. Throughput was maintained for large object counts where traditional systems perform inadequately. The area-efficient microarchitecture with a spatial-partitioning cell size of 1024 was found to be the optimal of our designs due to its greater exploitation of parallelism. Moreover, fixed-function microarchitectures significantly reduce power consumption, and this allowed us to mitigate silicon power-density challenges, while facilitating computation on mobile and wearable computing devices.

Although we have demonstrated that our system is highly efficient, there exists some scope for expansion. In particular, it would be interesting to experiment with shifting additional stages of the interactive application program loop from processors to microarchitectures. For instance, a microarchitecture for constructing BVHs [48] could complement ours. This would accept the *k*-d tree cells generated by the processor and construct or refit the BVHs associated with each cell. It

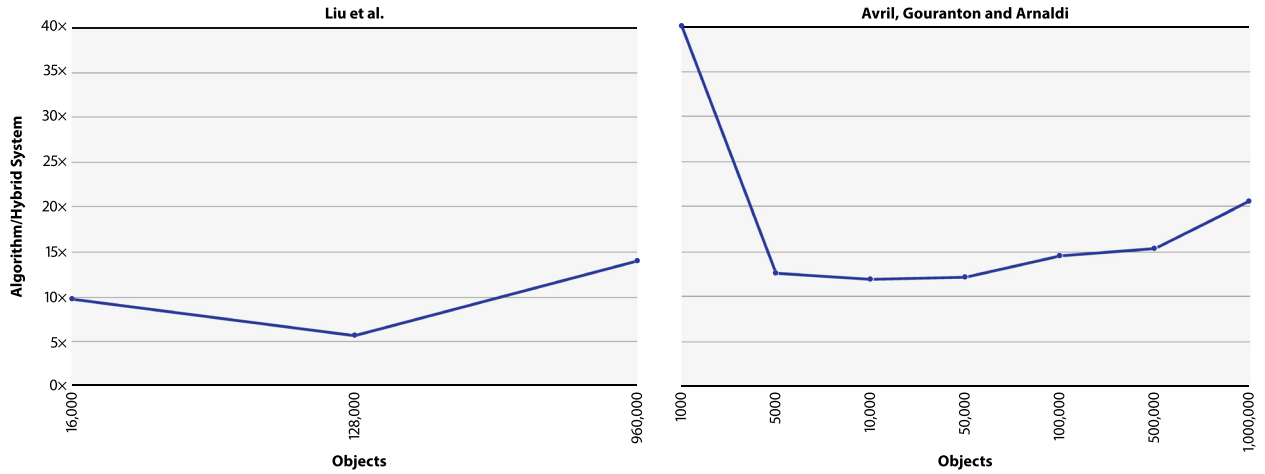

Fig. 5 Acceleration factors of the hybrid system using the area-efficient microarchitecture relative to recent GPU broad-phase collision-detection algorithms

would also be constructive to try adding an aforementioned narrow-phase microarchitecture [29, 30]. Overall, it is evident that these proposed ideas would develop our system, but, even without adaptation, our system achieves significant throughput and power efficiency advantages.

Abbreviations

AABB, axis-aligned bounding box; APU, accelerated processing unit; ASIC, application-specific integrated circuit; BVH, bounding-volume hierarchy; CAD, computer-aided design; DBVT, dynamic bounding-volume tree; FPGA, field-programmable gate array; HDL, hardware-description language; IC, integrated circuit; *k*-DOP, discrete-oriented polytope; OBB, oriented bounding box; SAH, surface-area heuristic; SAT, separating-axis test; VLIW, very-long instruction word

Competing interests

The authors declare that they have no competing interests.

Acknowledgements

The authors wish to express their gratitude to Michael J. Doyle for providing guidance on the ray-tracing elements of this article.

Funding

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

Received: 22 September 2015 Accepted: 7 June 2016
Published online: 27 June 2016

References

1. PM Hubbard, Approximating polyhedra with spheres for time-critical collision detection. ACM Trans. Graph. **15**(3), 179–210 (1996)
2. D Baraff, *Dynamic simulation of non-penetrating rigid bodies. PhD thesis*. (Cornell University, Ithaca, New York, USA, 1992)
3. JD Cohen, MC Lin, D Manocha, MK Ponamgi, in *Proceedings of the 1995 Symposium on Interactive 3D Graphics (i3D 1995)*. I-COLLIDE: An interactive and exact collision detection system for large-scale environments (ACM, New York City, New York, USA, 1995), pp. 189–196
4. S Gottschalk, MC Lin, D Manocha, in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 1996)*. OBBTree: A hierarchical structure for rapid interference detection (ACM, New York City, New York, USA, 1996), pp. 171–180
5. K Zikan, P Konečný, in *Proceedings of the 5th International Conference in Central Europe on Computer Graphics and Visualization (WSCG 1997)*. Lower bound of distance in 3D (University of West Bohemia, Plzeň, Czechia, 1997), pp. 640–647
6. H Jin, Z Liu, T Wu, Y Wang, in *Proceedings of the 2009 International Conference on Computer Engineering and Technology (ICCET 2009)*. The research of collision-detection algorithm based on spatial subdivision (IEEE Computer Society, Los Alamitos, California, USA, 2009), pp. 452–455
7. G van den Bergen, Efficient collision detection of complex deformable models using AABB trees. J. Graph. Tools. **2**(4), 1–14 (1997)
8. M Teschner, S Kimmerle, B Heidelberger, G Zachmann, L Raghupathi, A Fuhrmann, M-P Cani, F Faure, N Magnenat-Thalmann, W Strasser, P Volino, Collision detection for deformable objects. Comput. Graph. Forum. **24**(1), 61–81 (2005)
9. Z Wang, M Tang, R Tong, D Manocha, TightCCD: Efficient and robust continuous collision detection using tight error bounds. Comput. Graph. Forum. **34**(7), 289–298 (2015)
10. L He, R Ortiz, A Enquobahrie, D Manocha, in *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games (i3D 2015)*. Interactive continuous collision detection for topology changing models using dynamic clustering (ACM, New York City, New York, USA, 2015), pp. 47–54
11. MC Lin, JF Canny, in *Proceedings of the 1991 IEEE International Conference on Robotics and Automation (ICRA 1991)*. A fast algorithm for incremental distance calculation (IEEE, New York City, New York, USA, 1991), pp. 1008–1014
12. B Mirtich, V-Clip: Fast and robust polyhedral collision detection. ACM Trans. Graph. **17**(3), 177–208 (1998)
13. NK Govindaraju, D Knott, N Jain, I Kabul, R Tamstorf, R Gayle, MC Lin, D Manocha, Interactive collision detection between deformable models using chromatic decomposition. ACM Trans. Graph. **24**(3), 991–999 (2005)
14. F Liu, T Harada, Y Lee, YJ Kim, Real-time collision culling of a million bodies on graphics processing units. ACM Trans. Graph. **29**(6), 154:1–154:8 (2010)
15. Q Avril, V Gouranton, B Arnaldi, in *Proceedings of the 2012 Eurographics Symposium on Parallel Graphics and Visualization (EGPGV 2012)*. Fast collision culling in large-scale environments using GPU mapping function (Eurographics, Aire-la-Ville, Switzerland, 2012), pp. 71–80
16. C Lauterbach, Q Mo, D Manocha, gProximity: Hierarchical GPU-based operations for collision and distance queries. Comput. Graph. Forum. **29**(2), 419–428 (2010)
17. M Tang, D Manocha, J Lin, R Tong, in *Proceedings of the 2011 Symposium on Interactive 3D Graphics and Games (i3D 2011)*. Collision-streams: Fast GPU-based collision detection for deformable models (ACM, New York City, New York, USA, 2011), pp. 63–70
18. X Zhang, YJ Kim, Scalable collision detection using *p*-partition fronts on many-core processors. IEEE Trans. Visual. Comput. Graph. **20**(3), 447–456 (2014)

19. D Kim, J-P Heo, J Huh, J Kim, S-e Yoon, HPCCD: Hybrid parallel continuous collision detection using CPUs and GPUs. Comput. Graph. Forum. **28**(7), 1791–1800 (2009)
20. I Wald, V Havran, in *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing (RT 2006)*. On building fast kd-trees for ray tracing, and on doing that in O(N log N) (IEEE Computer Society, Los Alamitos, California, USA, 2006), pp. 61–69
21. I Wald, S Boulos, P Shirley, Ray tracing deformable scenes using dynamic bounding volume hierarchies. ACM Trans. Graph. **26**(1), 16:1–16:18 (2007)
22. J Pantaleoni, D Luebke, in *Proceedings of the 2010 Conference on High-Performance Graphics (HPG 2010)*. HLBVH: Hierarchical LBVH construction for real-time ray tracing (Eurographics, Goslar, Germany, 2010), pp. 87–95
23. H Esmaeilzadeh, E Blem, R St. Amant, K Sankaralingam, D Burger, Dark silicon and the end of multicore scaling. ACM Trans. Graph. **39**(3), 365–376 (2011)
24. G Venkatesh, J Sampson, N Goulding, S Garcia, V Bryksin, J Lugo-Martinez, S Swanson, M Bedford Taylor, in *Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2010)*. Conservation cores: reducing the energy of mature computations (ACM, New York City, New York, USA, 2010), pp. 205–218
25. ES Chung, PA Milder, JC Hoe, K Mai, in *Proceedings of the 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2010)*. Single-chip heterogeneous computing: Does the future include custom logic, FPGAs, and GPGPUs? (IEEE Computer Society, Los Alamitos, California, USA, 2010), pp. 225–236
26. T Mitra, Heterogeneous multi-core architectures. IPSJ Trans. Syst. LSI Design Methodol. **8**, 51–62 (2015)
27. R Rajwar, M Dixon, R Singhal, in *Proceedings of the 7th Biennial Conference on Innovative Data Systems Research (CIDR 2015)*. Specialized evolution of the general-purpose CPU, (Asilomar, California, USA, 2015). http://cidrdb.org/cidr2015/
28. YS Shao, S Xi, V Srinivasan, G-Y Wei, D Brooks, in *Proceedings of the 2015 Sensors to Cloud Architectures Workshop (SCAW 2015)*. Toward cache-friendly hardware accelerators, (Bay Area, California, USA, 2015). http://darksilicon.org/hpca/SCAW-2015/
29. N Atay, JW Lockwood, B Bayazit, A collision detection chip on reconfigurable hardware. Technical Report WUCSE-2005-33, (Washington University, St. Louis, Missouri, USA 2005)
30. A Raabe, F Zavelberg, in *Proceedings of the 16th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG 2008) – Communication Papers*. Defying the memory bottleneck in hardware accelerated collision detection (UNION Agency-Science, Plzeň, Czechia, 2008), pp. 25–31
31. C Davis, M Hedge, OA Schmid, M Maher, JP Bordes, Physics Processing Unit. International Patent WO/2005/038559 (2005)
32. M Woulfe, J Dingliana, M Manzke, in *Proceedings of the 4th Workshop on Virtual Reality Interaction and Physical Simulation (VRIPHYS 2007)*. Hardware accelerated broad phase collision detection for realtime simulations (Eurographics, Aire-la-Ville, Switzerland, 2007), pp. 79–88
33. MC Lin, S Gottschalk, in *Proceedings of the 8th IMA Conference on the Mathematics of Surfaces*. Collision detection between geometric models: a survey (IMA, Southend-on-Sea, Essex, UK, 1998), pp. 37–56
34. W Fan, B Wang, J-C Paul, J Sun, A hierarchical grid based framework for fast collision detection. Comput. Graph. Forum. **30**(5), 1451–1459 (2011)
35. M Woulfe, M Manzke, in *Proceedings of the 25th Spring Conference on Computer Graphics (SCCG 2009)*. A framework for benchmarking interactive collision detection (Comenius University, Bratislava, Slovakia, 2009), pp. 221–228
36. A Beechick, S Casselman, L Yarbrough, Internal sorting and FPGA. Proc. SPIE. **2914**, 66–71 (1996)
37. S Wong, S Vassiliadis, JY Hur, in *Proceedings of the 16th Annual Workshop on Circuits, Systems and Signal Processing (ProRISC 2005)*. Parallel merge sort on a binary tree on-chip network (Technology Foundation STW, Utrecht, The Netherlands, 2005), pp. 365–368
38. J Harkins, T El-Ghazawi, E El-Araby, M Huang, in *Proceedings of the 2005 IEEE International Conference on Field-Programmable Technology (FPT 2005)*. Performance of sorting algorithms on the SRC 6 reconfigurable computer (IEEE, New York City, New York, USA, 2005), pp. 295–296
39. Y-K Chen, J Chhugani, CJ Hughes, D Kim, S Kumar, V Lee, A Lin, AD Nguyen, E Sifakis, M Smelyanskiy, High-performance physical simulations on next-generation architecture with many cores. Intel Technol. J. **11**(3), 251–261 (2007)
40. DJ Tracy, SR Buss, BM Woods, in *Proceedings of the 2009 IEEE Virtual Reality Conference (VR 2009)*. Efficient large-scale sweep and prune methods with AABB insertion and removal (IEEE Computer Society, Los Alamitos, California, USA, 2009), pp. 191–198
41. R Scrofano, L Zhuo, VK Prasanna, Area-efficient arithmetic expression evaluation using deeply pipelined floating-point cores. IEEE Trans. Very Large Scale Int. Syst. **16**(2), 167–176 (2008)
42. N Kurd, M Chowdhury, E Burton, TP Thomas, C Mozak, B Boswell, P Mosalikanti, M Neidengard, A Deval, A Khanna, N Chowdhury, R Rajwar, TM Wilson, R Kumar, Haswell: A family of IA 22 nm processors. IEEE J. Solid-State Circuits. **50**(1), 49–58 (2015)
43. B Munger, D Akeson, S Arekapudi, T Burd, HR Fair III, J Farrell, D Johnson, G Krishnan, H McIntyre, E McLellan, S Naffziger, R Schreiber, S Sundaram, J White, K Wilcox, Carrizo: A high performance, energy efficient 28 nm APU. IEEE J. Solid-State Circuits. **51**(1), 105–116 (2016)
44. M Joselli, J Ricardo da Silva Junior, M Zamith, E Clua, M Pelegrino, E Mendonça, E Soluri, in *Proceedings of the 4th International IEEE Consumer Electronics Society Games Innovation Conference (IGIC 2012)*. Techniques for designing GPGPU games (IEEE Computer Society, Los Alamitos, California, USA, 2012), pp. 51–55
45. S Laine, T Karras, in *Proceedings of the 2011 Conference on High-Performance Graphics (HPG 2011)*. High-performance software rasterization on GPUs (ACM, New York City, New York, USA, 2011), pp. 51–55
46. J Spjut, A Kensler, D Kopta, E Brunvand, TRaX: A multicore hardware architecture for real-time ray tracing. IEEE Trans. Computer-Aided Design Integr. Circuits Syst. **28**(12), 1802–1815 (2009)
47. J-H Nah, J-S Park, C Park, J-W Kim, Y-H Jung, W-C Park, T-D Han, T&I engine: Traversal and intersection engine for hardware accelerated ray tracing. ACM Trans. Graph. **30**(6), 160:1–160:10 (2011)
48. MJ Doyle, C Fowler, M Manzke, A hardware unit for fast SAH-optimised BVH construction. ACM Trans. Graph. **32**(4), 139:1–139:10 (2013)
49. I Kuon, J Rose, Measuring the gap between FPGAs and ASICs. IEEE Trans. Computer-Aided Design Integr. Circuits Syst. **26**(2), 203–215 (2007)
50. R Hameed, W Qadeer, M Wachs, O Azizi, A Solomatnikov, BC Lee, S Richardson, C Kozyrakis, M Horowitz, in *Proceedings of the 37th Annual International Symposium on Computer Architecture (ISCA 2010)*. Understanding sources of inefficiency in general-purpose chips (ACM, New York City, New York, USA, 2010), pp. 37–46