The paper describes a stream processing architecture for ray tracing. The architecture consists of two types of processing elements, one to handle address generation and one to handle computation.

A thorough evaluation is given ray-tracing three scenes. The evaluation explores the design space along several dimensions. One of the more interesting findings is that partitioning the address and data calculations as described results in a 56% speedup.

>>> Evaluation <<<

Strengths

The finding that partitioning address and data calculation in the manner described greatly improves performance is a novel and useful finding. However, more detail should be given both on the partitioned organization and on the methodology for arriving at this result.

The method of filtering appears to greatly improve efficiency of SIMD execution engines on ray tracing. However, this idea is nearly identical to "conditional streams" (Kapasi et al., Micro 2000) – which should be referenced and discussed in the related work section.

Weaknesses:

The paper is a bit off topic for ASPLOS. It may be a better match for SIGGRAPH or one of the Graphics Hardware Workshops.

The related work section needs a lot of work. First, the conditional streams paper (see above) needs to be compared to "stream filtering". Second this paper needs to be put in context with the Purcell et al. paper from SIGGRAPH 2002 – probably the closest piece of related work.

Finally, several of the papers that are listed in the references, but not discussed in the related work section need to be discussed and the differences and similarities with the proposed architecture described – in particular [8,11,14,17,20 and 30].

>>> Questions for the authors to address <<<

Please explain exactly the experiment run to conclude that partitioning data and address calculation results in a 56% speedup.

Please expand the related work section to describe the differences between this
work and (1) Kapasi et al. (Micro 2000), and references [8,11,14,17,20 and 30].

>>> Summary of the submission <<<

This paper describes a custom architecture aimed at improving SIMD throughput as applied to ray tracing kernels. The use of SIMD in ray tracing has been proven to greatly accelerate the algorithm as coherent packets of rays can traverse through the same paths in acceleration data structures and also in the scene geometry. The utilization of a SIMD register can falter though as different parts of the packet intersect separate parts of the geometry. The packet must however continue on until all of its constituent rays have terminated somewhere. This problem increases as SIMD registers grow in size and hence the number of rays that can be bundled into each register. These problems are highly apparent with secondary rays as they are by nature less coherent. The architecture and technique described (Stream Filtering) overcomes these limitations of SIMD based packet tracing.

>>> Evaluation <<<

This paper is generally good, but there are two weaknesses in the experimental evaluation, and the authors should say something about ray tracing of dynamic scenes using their architecture. The paper claims large speed increases and proposes a novel architecture. However, the datasets upon which the data to support the paper was generated from, are very few at only 3 scenes. Of these scenes, only 1 (the conference room) is a canonical scene commonly used in the real-time ray-tracing community. I would definitely like to see more scenes tested and a discussion, if necessary, of why the architecture performed better on some scenes and not others.

No mention is made of dynamic scenes. It is apparent from the paper that all scenes are of the simple static type. Is the architecture therefore limited to static scenes? The Real-Time Ray-Tracing of Dynamic Scenes on custom hardware has been demonstrated in "Realtime Ray Tracing of Dynamic Scenes on an FPGA Chip" (Jörg Schmittler, Sven Woop, Daniel Wagner, Wolfgang J. Paul, and Philipp Slusallek) at Graphics Hardware 2004.

The number of samples per pixel is set at 64. This is quite a large value and guarantees the rays generated for each pixel will be highly coherent. This may artificially raise SIMD utilization figures. I would like to see performance of the system with samples per pixel at lower figures such as 1,4,16,32 etc.

Minor typo in "Related work" under Breadth-First ray-tracing "...access to scene data and Smaximise".

>>> Questions for the authors to address <<<

This paper is generally good, but there are two weaknesses in the experimental evaluation, and the authors should say something about ray tracing of dynamic scenes using their architecture. The paper claims large speed increases and proposes a novel architecture. However, the datasets upon which the data to support the paper was generated from, are very few at only 3 scenes. Of these scenes, only 1 (the conference room) is a canonical scene commonly used in the real-time ray-tracing community. I would definitely like to see more scenes tested and a discussion, if necessary, of why the architecture performed better on some scenes and not others.

No mention is made of dynamic scenes. It is apparent from the paper that all
scenes are of the simple static type. Is the architecture therefore limited to static scenes? The Real-Time Ray-\(\text{Tracing of Dynamic Scenes on custom hardware has been demonstrated in "Realtime Ray Tracing of Dynamic Scenes on an FPGA Chip" (J\text{"o}rg Schmittler, Sven Woop, Daniel Wagner, Wolfgang J. Paul, and Philipp Slusallek) at Graphics Hardware 2004.}

The number of samples per pixel is set at 64. This is quite a large value and guarantees the rays generated for each pixel will be highly coherent. This may artificially raise SIMD utilization figures. I would like to see performance of the system with samples per pixel at lower figures such as 1,4,16,32 etc.

Minor typo in "Related work" under Breadth-First ray-tracing ":...access to scene data and 5maximise".

>>> Summary of the submission <<<

SUMMARY:

It is well-known that current z-buffer algorithms, designed for far fewer transistors than are available today, make graphics authors perform weird approximations to deal with the effects of light reflected off of objects. It is also well-known that ray-tracing naturally deals with effects like replication, but that it is too slow to do this in real time on conventional host hardware and that it is a poor match for conventional GPUs.

The present paper is an ASPLOS-appropriate effort of a software-hardware co-design for ray-tracing. The authors build on Bribble and Ramani methods of specifying ray-tracing in terms of filters on streams of rays. Their approach uses specialized address-generation units (AGUs) to gather stream elements from memory for dispatch at array of SIMD stream processors. The results show a high SIMD unit utilization, which, at least, shows that the system is balanced.

>>> Evaluation <<<

EVALUATION:

This is a bold ASPLOS paper, as we want to take papers that exploit software-hardware co-design for important applications such as high-quality real-time graphics.

The paper is lacking many details; however, this may be attributed, in part, to the fact that it is so ambitious. While I think that I understand ray tracing at some level, the code at the top of page two makes no sense to me. The internals of the hardware block get little explanation. How does the new language compare and contrast with MIT’s StreamIt (http://www.cag.lcs.mit.edu/streamit/)? The simulator used is barely described except to say "similar to SimpleScalar," which I can’t believe is true, since SimpleScalar only handles a single core system.

Future versions of this paper should compare with Intel’s Larrabee, Seiler, et al., SIGGRAPH, 2008 and Govindaraju, et al., “Toward A Multicore Architecture for Real-Time Ray-tracing,” MICRO 2008. Both of these papers, however, were not available until AFTER the ASPLOS 2009 submission date.

To supplement my only high-level knowledge, I solicited a co-review:

1) This paper is similar to raytracing on Cell.

http://graphics.cs.uni-sb.de/~benthin/cellrt06.pdf
Ray Tracing on the CELL processor
Carsten Benthin, Ingo Wald, Michael Scherbaum, and Heiko Friedrich
This work basically also uses BVH traversal, double-buffering, and intelligent b/w management to do raytracing on Cell. Obviously the difference is N, 4-wide cores as opposed to their design of one large N-wide core. This paper does not cite the CELL paper which I think they should.

2) Table 4 data to me is a little odd. I would expect intersection to show the most SIMD utilization. The paper doesn't explain why the SIMD utilization is so low for intersection.

Methodology:
*) The memory modeling or the lack of it is a big hole to me.

*) I have concerns about what they are simulating. Surely they cannot be simulating the full scene! And I don't understand their definition of samples per pixel. Is this same as primary rays per pixel?

*) Their numbers don't seem to add up. There is a high chance I got this math wrong, but bear with me! Table 3 says, per frame # rays = 1.xx * 10^8. Assume conservative 10 instructions are required per ray. That gives a total of 10 * 10^8 instr. per frame. For an FPS for 20 fps, that is a total of 20 billion inst/second. At a frequency of 500 MHz, this is a total of 40 inst/cycle. The largest SIMD width is 16, so I am not sure what is going on.

*) Also I don't exactly understand what the traversal ops and intersection ops means in table 3. Are these individual operations counts in the ISA of the machine?

*) And the biggest hole for me is that this work does not do dynamic scenes which are much more important. As far as I can tell the BVH data structures are built once, or they assume they are built on the host PC. What happens if this data structure has to be modified when the scene changes? It is not clear that is feasible at real-time rate. They should say how it is feasible.

*) I am obviously missing something here. Do they have many x-wide units (where x=8,12,16 in table 5); Are they getting real-time rates at 500 MHz with one 16-wide unit? Cell has eight 4-wide SIMD (effectively twice the computation power) units running at 3.6 GHz and gets close to real-time for complex scenes. From this paper, I don't see where their 14X advantage over Cell comes from. (3.6/500 MHz * 32/16).

See questions from the co-review.

Fourth reviewer's review

>>> Questions for the authors to address <<<

This paper presents an efficient architecture for stream filtering. The main characteristics are isolation of data and address processing and use of SIMD processing. The paper is very well written and the analysis is quite extensive.

>>> Evaluation <<<

1. This paper is an extension of the authors’ prior work (ref 9). It is not clear from the description where work in ref 9 ends and this paper begins. My assumption (since I have not read ref 9) is that this is an implementation of the stream filtering approach presented in ref 9. The distinction needs to be made clearer.

2. StreamRay architecture: In the filter engine shown in Fig 2, are there two types of SIMD cores? There is a dotted line between one set that is confusing.
In the same note, in Fig 3, there is a dotted line between a subset of the AGUs and the corresponding banks. You say that the execution units process operands in SIMD or scalar fashion. How many operands does each execution unit operate on at a time?

3. Ray engine: The memory size for a 64x64 ray is given. Is this ray size based on the three test cases that you considered? How representative are these test cases? How scalable is this architecture to different ray sizes?

4. SIMD width: In the experiments, SIMD widths of 8, 12 and 16 were considered. The SIMD utilizations listed in table 4 for all three cases are comparable. Was there any reason why larger SIMD widths were not considered? In the Introduction (page 3), you say that ray tracing has been demonstrated to have over 91% efficiency on 512 processors. So why did you restrict yourself to N <=16?

5. Performance: You say that a 12-wide SIMD machine is the optimum choice in this case study. Why? The number of frames per second is higher for N=16. Isn’t the overhead of address computation taken into account in the derivation of numbers for the performance analysis presented in Table 5?

StreamRay efficiency: You seem to suggest that this architecture is unique in that the address and data computations are done separately. But such a paradigm has been used for a while in the design of programmable processors in digital signal processing, image processing, video processing and more recently wireless communication algorithms in software defined radios. On page 4 of Introduction you give the impression that there are other architectures for ray processing that perform data and address computation units on a shared set of EUs. What are these architectures?

>>> Questions for the authors to address <<<

1. Is this paper an implementation of the algorithm presented in ref 9? Please clarify.
2. Why did you choose SIMD widths <=16? Is the parallelism limited to 16?
3. Are there other ray tracing architectures? How does this compare to the existing ones?
Responses consisting of more than 500 words will not be stored!

In case of problems, please contact Richard van de Stadt.