Two Schemes to Improve the Performance of a *Sort-Last* 3D Parallel Rendering Machine with Texture Caches

Alexis Vartanian, Jean-Luc Béchennec, and Nathalie Drach-Temam

LRI, Université Paris-Sud,
91405 Orsay France,
(alex,jlb,drach)@lri.fr
tel: 33 1 69 15 42 22
fax: 33 1 69 15 65 86

Abstract. A *sort-last* 3D parallel rendering machine distributes the triangles to draw to different processors. When building such a machine with each processor having a texture cache, the texture locality is worse and the performance is reduced. This article investigates two schemes to preserve this locality while keeping a good load balancing: triangle slicing and locality aware triangle distribution. With both schemes, the speedups are improved between 2 and 6 times.
Keywords: Cache memories, multiprocessing, application specific architecture, parallel rendering, texture mapping.

1 Introduction

Multimedia applications are now expected to be the main consumer of computing power in the coming years. Among these applications, real-time 3D image rendering has raised a lot of attention this year as most microprocessor manufacturers are releasing new instruction set extensions (Intel SSE) mainly aimed at accelerating geometry transformations of PC microprocessors. Such a trend reduces the performance gap between a high end PC with a 3D accelerator and a workstation with expensive ASICs. As 3D rendering contains a fair amount of parallelism, a parallel machine with PC 3D accelerators could offer a good performance/price ratio.

There are many ways to use parallelism in 3D rendering. Molnar have given a classification of all kinds of machines. In this paper, according to this classification, we focus on a *sort-last* machine based on PC 3D accelerators. In such a machine, the triangles to be drawn are distributed among independent pipelines that do both geometry transformations and texture mapping on the same data. The PixelFlow machine is the best known sort-last 3D computer. This architecture has two main advantages: it is not necessary to redistribute the triangles between the geometry stage and the texture mapping stage and dynamic load balancing can be used. However these architectures have a main drawback: as each pipeline can render anywhere on the screen, all processors need to transfer

P. Amestoy et al. (Eds.): Euro-Par'99, LNCS 1685, pp. 757–760, 1999.

their frame buffer to one processor. This part of the machine requires a high bandwidth.

This is the reason why most research on sort-last machines has focused on this part of the pipeline. However, when such a machine is based on PC 3D accelerators, they will have an internal texture cache. Hakura and Gupta have shown that a cache works very well in handling texture locality. But when texture caches are used in a 3D parallel machine, a problem arises. If two triangles are rendered in different pipelines and if some *texels* [1] were reused between the two triangles, there could be less locality on a parallel machine compared to a sequential machine. In our previous work, we have observed such a phenomenon. We have shown that a synchronous *sort-middle* architecture performance was strongly reduced if such a phenomenon was taken into account. We have also evaluated sort-last machines and shown that some benchmarks could suffer from this phenomenon.

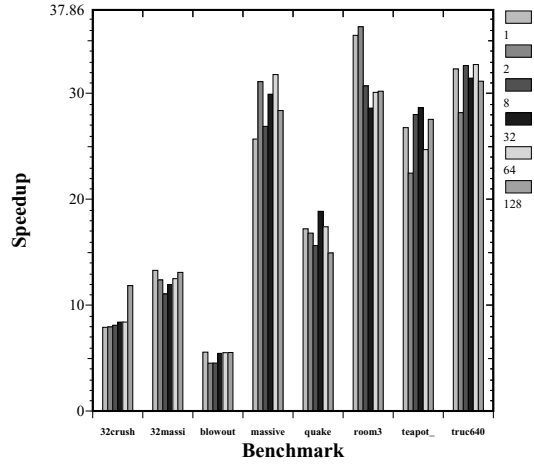

Fig. 1. *Speedup compared to a single processor machine with triangle set size, with a 64 processor machine having texture caches of 16KB.*

In this study, we evaluate two schemes to deal with this problem. These evaluations are based on event-driven simulations of the architecture on different 3D benchmarks. The first section presents the performance of the machine with the second scheme: we try to find out a distribution of the triangle that does not break the locality and we show that it generates load unbalance. The second section shows how we solve that problem with triangle slicing. The third section presents the overall performance when both schemes are used [2].

[1] A texel is a pixel in the texture.

[2] An more detailed version of this paper is available at http://www.lri.fr/∼alex/.

2 The Impact of Load Unbalance

The main idea of our whole study is to find out a triangle distribution algorithm that improves the locality. The algorithm we use in this study is the following: each time a geometry engine has finished its work, it asks the distributor for a set of N consecutive (in the submission order) triangles. With this algorithm, we expect to increase the cache locality. However if we try to use this algorithm alone, we don't success in increasing the performance as seen in figure 1. We see that speedup is not increasing with N. Moreover, the best value for N is not the same for all the benchmarks. It means that when this algorithm is used, it is not clear whether it increases or decreases the performance. This is due to the fact that the load balancing heavily depends on the big triangles. And our algorithm might send a set of very big triangles to the same processor. We are going to present a solution to this issue.

3 Triangle Slicing

The suggested scheme is the following. Each time a triangle has more pixels than S, the distributor cuts it into two fair slices. It does that while any triangle is bigger than S. We have measured the impact of such a scheme on load balancing with a perfect cache. If we cut any triangle having more than 2500 pixels, the negative impact of the first scheme disappears and the speedup with 64 processors is the same with any set size.

However, such a scheme has an impact on the locality. When a big triangle is sliced, two effects are expected. The locality might be improved: as triangles are smaller, the lines could be shorter. If a texel is shared between two pixels on the same row, the duration between the two uses depends on the line size. If the line is too long, the texels might have been evicted from the cache because of a conflict or a capacity problem. On the other hand, when a triangle is sliced, the pixels of the side could have less locality. As far as these pixels are concerned, the duration between the texel uses is far longer with slicing. Before slicing, these texels were immediately reused. With slicing, they have to wait for the complete triangle to be drawn.

We observed that the impact of the two phenomena were highly dependent on the slicing method used. When a triangle is cut into two triangles, we have to choose on which side we add a vertex. We evaluated different algorithms and showed that it is better to choose the highest side than the biggest side. It generates less high triangles than the other. If the triangles are less high with the same size, they must have a greater width. As we said before, this is an important issue for the locality.

4 Evaluation of the Distribution Scheme

We now measure the performance of the architecture with triangle slicing. We observe on figure 2 that, for all the benchmarks, performance increases with N.

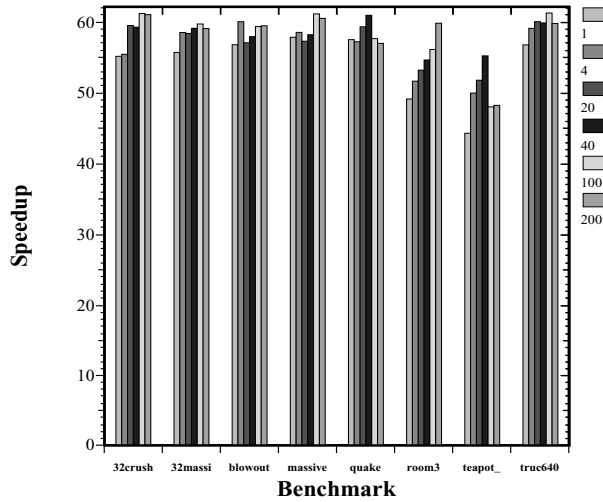

Fig. 2. *Speedup compared to a single processor machine with triangle set size, with a 64 processor machine having texture caches of 16KB. Triangle are sliced if bigger than 1000 pixels with the algorithm 2.*

This is what we expected: the locality is improved by our algorithm. However for *quake* and *teapot.full* performance starts to decrease if N is greater than 40. To understand that phenomenon we measured the average cache miss rate of all the processors. We saw that for all the benchmarks, the miss rate decreases between 1 and 40 and increases with N after 40. We have the following analysis of this behavior: if too many triangles are sent, the number of triangles to be redistributed at the end of the frame increases. The impact of this on performance depends on the bus load at this period. It shows that if we want to have the optimal performance with any benchmark, a set size of 40 should be used.

5 Conclusion

In this study, we have seen that the performance of a sort-last parallel machine aimed at real-time 3D rendering and based on PC 3D accelerators greatly depends on texture cache behavior and on load balancing. We have seen that any attempt to manage the locality with a good distribution was limited by the big triangles. We have presented a solution to this problem: triangle slicing. However, if this is done, locality can be greatly reduced. To avoid this behavior, we have suggested a slicing algorithm that keeps a good locality. With that slicing algorithm, we have shown that if the distributor sends 40 triangles to a pipeline each time it is free, the locality is increased and the performance is better. With these two schemes, we are able to increase the speedup between 2 and 6 times.

This paper is a summary of our work. A more detailed version with complete references is available at **http://www.lri.fr/∼alex/**.