Analyzing Data Reusability of Raytrace Application in Splash2 Benchmark

Hao Do-Duc[1,2], Vinh Ngo-Quang[3]
[1]Division of Computational Mathematics and Engineering (CME), Institute for Computational Science (INCOS), Ton Duc Thang University, Ho Chi Minh City, Vietnam
[2]Faculty of Information Technology, Ton Duc Thang University, Ho Chi Minh City, Vietnam
[3]IC Design Research and Education Center, Vietnam National University, Ho Chi Minh City, Vietnam

***Abstract*—[1]When designing a chip multiprocessors, we use Splash2 to estimate its performance. This benchmark contains eleven applications. The performance when running them is similar, except Raytrace. We analyse it to clarity why the performance is not good. We discover, in theory, Raytrace never reuses data. This leads the fact that the performance is not good due to the low hit ratio in data cache.**

Keywords—Chip multiprocessors; benchmark; ray tracing; reflection; intensity; ray-Tree

I. INTRODUCTION

When designing Chip Multi-Processors (CMP), we always use one or many benchmarks to evaluate our products. One of the most used benchmark is Splash2 [1]. It contains many applications such as: FFT (Fast Fourier Transform) [2], Cholesky factorization [3], Barnes (N-Body problem) [4], etc. They are the most popular classical problems in parallel computing, the main applied field of CMP. Each of these problems requires its unique kind of data and the way to solve it. The complexity problems in real life, in general, are the combination of some basic problems, which are included in Splash2. If a CMP solves the basic problem well, it will solve the real problem well too.

Many CMPs are regularly used to process computer graphics. For this, Splash2 provides three relevant applications namely Radiosity, Raytrace, and Volrend. In many experiments, however, the performance when running Raytrace is not good while the others are better. This inspires us to study and analyze Raytrace application and explain why the CMP do not solve it well.

Our paper is organized as follows. It begins with the introduction of CMP benchmark and questions why the performance of CMP when running Raytrace is not good. Section II presents clearly about Raytrace applications. This section begins with the rendering problem in computer graphics and analyzes the ray tracing method after that. Section III shows us how to use parallel computing to do ray tracing method. The experiments are presented in section IV. We run Raytrace in many CMPs to evaluate the performance, and then, we show its performance in comparison with other applications. The final section is the conclusion. It is presented in section V.

[1]This work was supported by Vietnam National University - Ho Chi Minh city grant number C2015-40-01.

II. RENDERING IN COMPUTER GRAPHICS

We are living in a 3-Dimension (3D) space, but our eyes only observe 2-Dimension (2D) of the world. How we impress the real world by observation? That is based on our hobbies; we can change the view points to get more information about the locations of many objects. So that, we can image exactly where an object is. But we can not change our viewpoint when using a monitor such as a computer monitor. From a 2D image in the monitor, how can we identify the location of any object? That is up to the way we present the image in the monitor. How we present a 2D image, which helps us to impress the location of each object, is called rendering.

A. What is rendering?

Rendering is the way to present a 2D image, which helps us to image about its 3D sense or the location of each object. An object in 3D space is identified by three information: height, width, and depth. 2D image presents the height and the width, and the rendering problem presents the depth of the object. There are two main types of rendering: local illumination and global illumination. Both of them use the intensity, which is from the light, to present the deep of each object. But they use the lights in different ways. The local illumination is very simple. We only use the light coming directly from a light source for the image. That means we do not use others kind of light such as light reflected from a mirror to present the object. Its advantage is simple in both idea and coding, but it is not really a good method. Global illumination is more complexity and efficient than local illumination. We consider all of the lights while presenting an object: directed light, reflected light and shadow light. From these lights, we can create many effects such as reflection, shadows. This approach is the main method for this problem in modern graphics.

B. Raytrace method

Global illumination contains many methods such as ray casting, ray tracing, etc. They use many kinds of light to present an object. Ray tracing is popularly used in both industry and personal applications. Its idea is simple: tracing a path from a point of view through each pixel in a virtual screen, then calculating the light intensity and the color of the object which is visible through it[5]. First, we need to define the main problem: we have a set of light sources and a set of objects, and their location in 3D space. We want

to compute the intensity and colors of each pixel, which is used to present the 3D space including the mentioned light sources and objects, on the screen. Figure 1 shows us an illustration for ray tracing

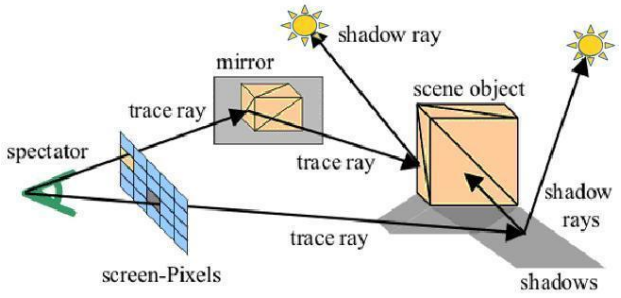

Fig. 1. Identifying the value for every pixel using ray tracing. (Source: ICE RWTH Aachen University)

We want to compute the value at the pixel in the intersection point between the line, that connects our eye and the object, and the screen. Besides, the connecting line contains not only the directed light but also the reflected light from the other surfaces. This leads us that we can observe both the scene in the viewport and the scene which is reflected by the objects in the viewport. Imaging, from our eye, a ray is released. It meets a surface and is reflected. In the real world, almost every object do not have a pure smooth surface, so the reflected rays are spread or diffuse like the illustration in figure 2.

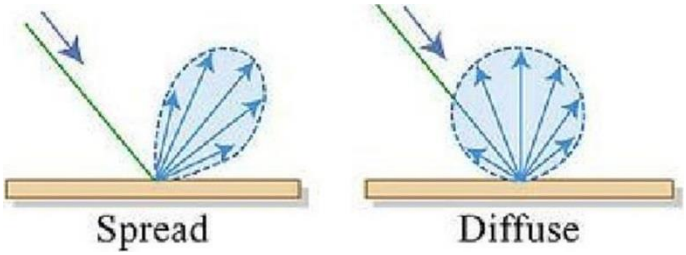

Fig. 2. When a ray meets a non-pure smooth surface, we receive the spread or diffuse reflection. (Source:MIT Open-CourseWare)

The corollary of spreading or diffusing reflection is that the reflected rays will meet many objects, and that process will repeat. But there is an important note: the power of a light ray is reduced after each reflecting point. In other words, we can say there are many rays from many objects have the contribution to the value of a pixel, the less reflecting time, the more contribution. When considering the ray from our point of view to a pixel on the monitor, we need to compute a group of rays reflecting between many objects. If we choose a sequence of objects and identify the reflecting ray between them, we will receive a ray path. The destination of a ray path is often the light source. An object can also be a destination when a ray reflects many times and ends up at that object. Fig.3 shows us a ray path as an example. From the view point to the bulb-light source, the path connects three other objects. Figure 4 shows us a ray-tree [6] or a group of ray paths when extending the contribution to one pixel.

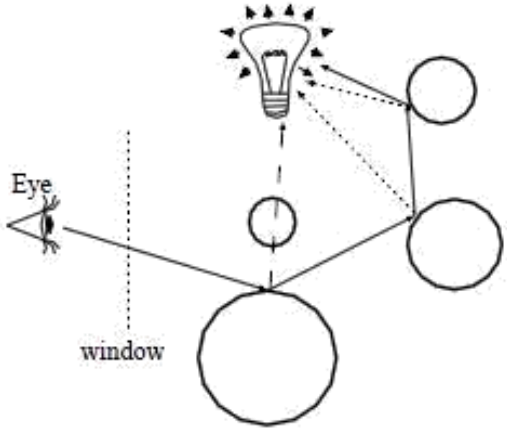

Fig. 3. One ray path in Ray tracing process

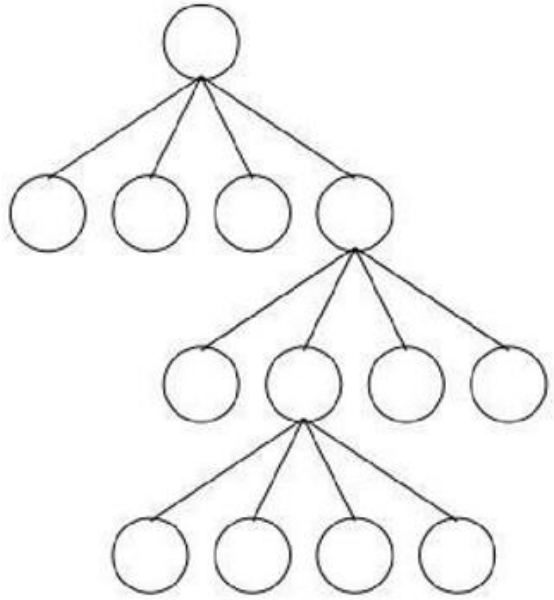

Fig. 4. The ray-tree when extending all of the contribution to one pixel from all objects

The color and intensity of one pixel are decided by one specific point of one specific object, and the value of that specific point is decided by many other points from many other objects. This loop is stopped when the number of reflections is large enough or, in other words, the contribution of a point or an object to the value of the pixel is small enough. We can use a hierarchical tree to illustrate this.

III. Using Parallel Computing to Solve the Rendering Problem by Ray Tracing Method

We presented the Ray tracing method in section II. Raytrace is the parallel version of that method. Our expected output is the color and intensity of all pixels on the screen. These value of one pixel are computed based on the pixel's ray-tree. In parallel method, multiple processor cores can

compute the value of multiple pixels simultaneously [6]. Because of our purpose, we analyze the property of the input data for Raytrace applications. The data is a set of light sources and objects. While calculating the value of a pixel through its ray-tree, an object can contribute different values for the computing process at different time because of the reflection. So we can consider their contributions are from different objects. On the other hand, if one object exists in two ray-trees or two pixels, its contributing values are neither the same because the view angles from two different pixels to the same object are different. Thus, we reach a conclusion that all nodes serving ray tracing process are not reused. This means that each node from each ray-tree is used just one time during their life. So, in theory, we can not reuse any node or any data for our computation.

IV. Performance of Cmp when Running Raytrace Application

This section presents two experiments focus on L1-Data cache. In the first experiment, we run a CMP using Raytrace and three other random selected applications as the workload. We will show the performance of CMP for each application in comparison with the others. In the second experiment, we show the performance of different CMP configurations when running Raytrace. This demonstrates that the negative properties of Raytrace are caused by theory, and they can not be solved by changing CMP. In this section, we use hit ratio in L1-Data cache as the measurement for estimating CMP's performance. This information is an important parameter of a CMP.

A. Experiment 1

We use three random applications in Splash2 to compare with Raytrace. The results are shown in figure 5. As we can see, in 4 cores the hit Ratios at L1-Data cache of Raytrace are significantly lower in comparison with the others. Its hit ratios are high, over 70%, because of the technique of coding. Three others have more positive properties, so the data is reused efficiently, and the hit ratio is nearly 99%, obviously higher than Raytrace.

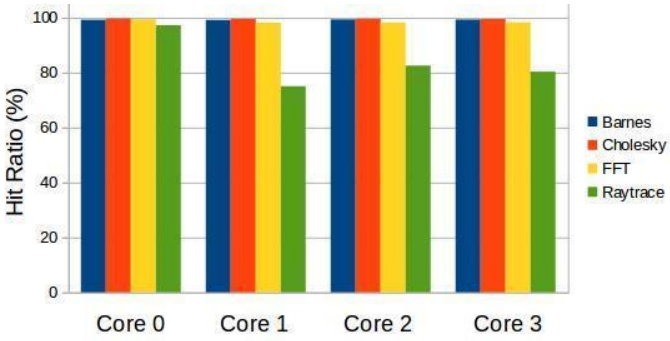

Fig. 5. Hit Ratio in L1-Data cache of CMP when running four applications

B. Experiment 2

We run Raytrace in 4 CMPs to estimate the average performance of reusing data. Our CMPs contain four processor cores. The sizes of L1-Data cache for the four CMPs are 4 KB, 16 KB, 64 KB and 256 KB, respectively. The results are presented in figure 6. When the cache size is too large, total 1 MB for L1-Data cache, the hit ratio is not high, just over 80%. With this result, we infer that the L1 data cache hit ratio or the performance of CMP can not be improved by increasing L1 cache size. We need to change the method or approach instead of changing CMP.

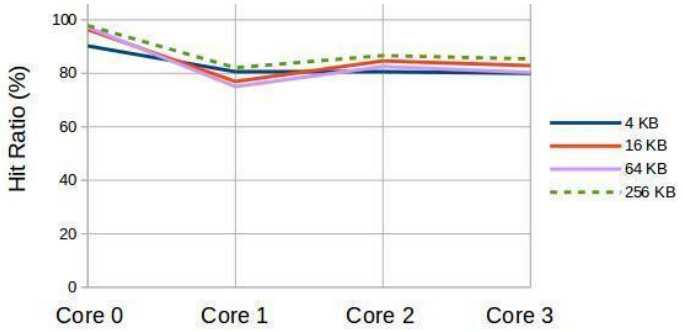

Fig. 6. Hit Ratio in L1-Data cache of 4 CMPs when running Raytrace

V. Conclusion

Our analysis proves that Ray tracing is a good method for the effects used in applications, but it is not well-fit for parallel computing hardware. Because the data is not reused, and each node is used just one time. This led the hit ratio in L1-Data cache is too low, and the performance is not good. We need a new parallel algorithm for this problem instead of increasing L1 cache size of the CMP.

References

[1] Steven Cameron Woo et al. "The SPLASH-2 Programs: Characterization and Methodological Considerations". 22nd Annual International Symposium on Computer Architecture. 1995

[2] Somasundaram Meiyappan. "Implementation and performance evaluation of parallel FFT algorithms". Technical report. 14 pp

[3] Rothberg and Gupta. "An efficient block-oriented approach to parallel sparse Cholesky factorization". ACM. 1993

[4] Jaswinder Pal Singh, John L. Hennessy and Anoop Gupta. "Implications of Hierarchical N-body Methods for Multiprocessor Architecture". ACM Transactions on Computer Systems. Pages 141-202. 1995

[5] Sid Delmar Leach. "3D Rendering". 224 pp. 2011.

[6] J. P. Singh et al. "Parallel Visualization Algorithms: Performance and Architectural Implications". 45-55. Journal of Computer. 1994