Splay Thread Cooperation on Ray Tracing as A Load Balancing Technique In Speculative Parallelism And Gpgpu

Suma Shivaraju[1] and Gopalan Pudur [2]
[1]Research Scholar, Bharathiar University, Coimbatore, India
[2]Professor, National Institute of Technology, Tiruchirapally, India

Abstract: *The introduction of the speculative parallelism into any models can improve the performance and provides significant benefits and increases the ILP and TLP. GPGPU is the future computing technology working with both CPU and GPU to solve many real-world problems not only the graphics problems but also the general purpose applications. As GPU uses data parallelism tasks, the dynamic memory creation and the splay trees which are self-adjusting allows for the increase in throughput and load balancing. The frequently used nodes near to the root are an advantage for finding locality of threads as well as for caching and garbage collection. A technique used to render and to study complex scenes into images and to render color, intensity of pixels, distance between pixels is referred as Ray tracing. Multithreading is a promising technique which increases the performance of the computer systems by increasing the instruction level parallelism and thread level speculation. In this paper a new technique is proposed for workload balancing on the Graphics processors and CPU that can be implemented on the graphics processors along with the CPU which provides the optimal result with the speculation techniques and Lorentz Transformation, which is used to determine color and brightness of the ray which are refracted or reflected and also the relative distance between the thread spanning which results in time dilation and contraction. A GPUOCELOT is a compilation framework, a simulator used for the execution of the programs which has resulted in the increase in the performance of the instructions which uses the amortized cost*

1. Introduction

Increase in the clock rate, exploiting the ILP through pipelining, out-of-order execution and multithreading are the techniques which results in reducing the dependencies and increase in the efficiency and performance of the microprocessors. Speculation techniques are implemented in the modern micro architectures to improve performance and to utilize the computer resources as well as the scheduling of the instructions based on the optimization techniques [1, 7]. The general purpose computing on graphics processing units handles the computations executed on the graphics processor or in kernels. The graphics processing units are computing on the data parallelism with fast access to memory and high throughput on parallel tasks. Since CPU's performs well on task parallelism and GPU's performs well on data parallelism. The memory systems are designed to stream data when the pattern can be accessed linearly and that can be prefetched [6, 14].

The combined technology of speculative execution on graphics processors still increases the performance and efficiency of the execution of the programs. Jaikrishnan Menon, Marc de Kruijf, Karthikeyan Sankaralingam proposed iGPU architecture executing the idempotent

regions Load balancing is a method for distributing the workloads across the processors [10].

The sharing of the workload across CPU and GPU needs to be balanced as the architectural designs of the CPU and GPU differs. Christian Lauterback, Qi Mo, Dinesh Manocha proposed explicit balancing on the GPU's and dynamic work Distribution [9].

In this model the hybrid architecture of CPU and GPU together holds threads of processing units and CPU does the task parallelism and GPU does the data parallelism [4, 13]. Distributing work across GPU and CPU processor's is an intellectual task so that the performance, scalability and tolerance factors of the processor is reduced instead of increase in the processor performance in ILP and TLP. Many Researchers' worked on the parallel ray tracing on the hybrid architectures with thread level speculation. Architectures are proposed to increase the performance through dynamic load balancing, reducing communication time, and reducing memory latencies, predicting value reuse, speculative computation reuse techniques between the CPU and GPU. But the overall speedup of computation, performance considerably is the linear speedup and the overheads incurred, granularity of the code

presented a major focus of issue. GPU supports fine grained parallelism over the data but the CPU works on the Coarse grained parallelism as the tasks. Transferring of the data and control values from CPU to GPU memory is challenging. Though with the speculative parallelization the state of the system is not altered, if the speculation fails, the normal execution of the task is carried out by the nonspeculative thread to recover the system state to safe.

To efficiently distribute the workload across the graphics processors a new technique is proposed which is a hybrid of CPU and GPU. A self adjusting splay tree provides load balancing on the GPU.

2. Splay Trees

Splay trees are the data structures where the binary search trees are self adjusting. Whenever we access a node of the tree, whether for retrieval, insertion or deletion a newly accessed node becomes the root of the modified tree. In splay trees the nodes which are frequently accessed move towards the root whereas the infrequent ones moves far away from the root. The different splaying steps which use the bottom-up approach are similar to the rebalancing operations on AVL & Red-Black trees. The cases for the splaying are ZIG-ZIG and ZIG-ZAG. In Zig-Zig, the splaying a node X which is a inside grandchild and it is a double rotation. In Zig-Zag, the splaying a node X which is a outside grandchild Where the grandchild is pulled to become the root of the sub tree [12].

2.1. Speculative Work Load Balancing

In this paper an algorithm is proposed where splaying steps happens for the threads in a block. The architecture of the speculatively parallelized graphics processors where the computations which are massively parallel in nature can be executed on both CPU and GPU, where the nonspeculative thread starts executing tasks on the CPU. The speculative threads execute tasks on the GPU. Another technique is proposed in this paper to balance the work load of the threads on the graphic processors.

The locality of reference to the code blocks on the graphic processors is scheduled using an algorithm where each thread of the block of a graphic processor is represented in the form of the nodes on a binary search tree. The splaying steps are applied on the tree of nodes such that the threads which are active and processing the computations are rotated near to the root node of the tree, which are always accessed frequently.

The different types of threads are running to balance the work loadi.e., some threads are in

a. Active state.
b. Passive state.
c. Squashed state.
d. Misspeculation state.

Some threads are at the initialization state. Some threads are waiting for some event.

The threads in the block are checked for the status. The different states the nodes are determined as nodes which are active, passive, nodes under initialization & miss peculation nodes.

2.2. Splaying on GPU's

On the graphic processor, the splaying operation is carried out, where root node is the nonspeculative thread and other child nodes are speculative threads. We apply splaying steps on these threads and determine whether the node is in what state i.e., threads on the GPU. Initially the current thread accessed is its states are determined based on the priority.

If the thread is active, its priority is high and that thread is right rotated and made as the child node of the nonspeculative thread. Successive access to a node newly increases the efficiency and locality of referencing to the nodes becomes easier. If the status of the node is determined, based on the status of the node the active nodes are moved towards the node and the nodes which are miss peculated/passive are moved away from the root node. Only active nodes become the children of the nonspeculative thread such that the load can be balanced between active nodes. Other nodes are squashed or eliminated from the binary search tree.
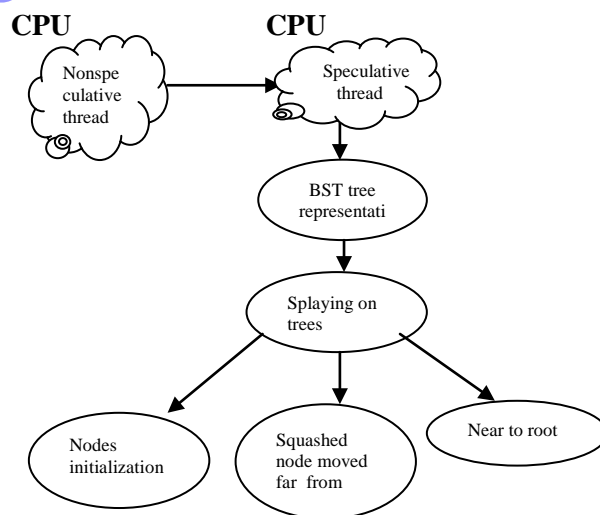

Figure 1. Diagrammatic representation of splaying operation on Threads on GPU.

Figure 1 describes the steps such that the state of the thread is determined based its locality of reference after splaying.

2.3. Design Methodology

Algorithm 1: The general algorithm for the splaying data structure.

Algorithm SplaySpec ()
This algorithm works on the CPU.

Step 1. Initialize the structure which holds elements No, left, right and the size of the nodes on the tree representation where each node represents the threads on the GPU.
Step 2. Function to create treeofthreads ()
Step 3. If tree is empty,
Step 4.Create the threadsoftree.
Step 5.Print the tree
Step 6. Print the each node of the thread of trees.

Algorithm 2: The Algorithm for the splaying operations.

```
#include<stdio.h>
#include<cuda.h>
# define NUM_BLOCKS 20;
```

Step 1. Structure for the tree.
Typedef struct splay_node splay;
Struct splay node {
                    *Splay *r,*l;*
                       Int info;
                  }
*_device_int * ptr [NUM_BLOCKS];*
Step 2. Memory Allocation.
_global void allocate memory ()
{
If (threadID.x==0)
*ptr [blockIdx.x]=(int *) malloc(blockDim.x*4);*
-syncthreads ();
If (ptr[blockIdx.x]==NULL)
Return;
ptr[blockIdx.x][threadIdx]=0
}
_global void memalloc()
{
  Int ptr1=ptr[blockIDx.x];
  If (ptr 1!=NULL)
  Ptr1(threadIDX.X] +=threadIDX.X
}
Step 3. Function splay tree manipulation.
*Tree *splay (int i ,tree *s)*
{
     *Tree n,*l,*r,*y;*
If (t==NULL) return t;
For (;;)
{
   If (i<t->info)
      { if (2t+1)==NULL) break;
        If (i < (2t+1).info)
           { z=2t+1
              2t+1=2t+2
              2t+2=z
              t=z
           }
        If(2t+1)==NULL) break;
}
    r.2t+1=t
    r=t
    t=2t+1
} else if (i > t.info)
{ if ((2t+2)==NULL) break ;
   If (i >2t+2.info)
{ z=2t+2
   2t+2=2t+1
2t+1=t
t=z
}
If ((2t+2)==NULL) break;

l.2t+2=t
l=t
t=2t+2

Else
{break ;}}
l.2t+2=2t+1
2t+1=2t+2
2t+2=n.2t+1
2t+1=m.2t+2
Return t
}

Initially, the memory is allocated dynamically using the functions allocatememory () and memalloc () functions [3]. The function splay() uses this array's and does the splaying by swapping the left child nodes to the right to balance the tree. Since the implementation of linked lists concept in GPU is little difficult as the memory allocation becomes difficult, a proposal to use the dynamic array is implemented for the splaying operation. Dynamic arrays are the growable arrays consisting of variable sized data structure where the addition and deletion happens and supported by the many of the modern programming languages.

The dynamic arrays are constructed using the geometric expansion of doubling in the size of the fixed array while it is used for reversed use such that n elements take $O(n)$ time such that each insertion takes amortized constant time.

These types of dynamic arrays are supported by the GPU's also. CUDA supports dynamic allocation of memory for a thread or a block. The performance issue relates to the dynamic array are that it includes Locality of reference and data cache utilization and it is random access also. The dynamic arrays supports for the faster indexing also [3]. Since the Amortized cost definitely guaranteed for the worst case analysis and performance than the speculation of the states in the program structure.

2.4. Splaying Transformations

The Figure 3 below shows the transformations on the splaying rotations of the threads in zigzag and zig-zig.Consider the threads on the GPU, each of the blocks of the threads holds 8 small threads inside numbered from 0 to 7 and each thread is identified by a threadID. The computations are allocated to the block of threads instead of individual threads on the block. Each block's threads are referred by the ThreadIDs and Blocks are represented by the BlockID's. BlockID=0 BlockID=1 BlockID=20 to 7 ThreadID's.

Suppose a computation is assigned to the BlockID=0, There are 7 threads whose synchronization is carried out using splaying zig-zag or zig-zig rotations as these threads are represented in the binary search tree.

The Zig-Zig rotation is carried out for checking the Activeness, passiveness, initialization of nodes and misspeculated nodes. Consider BlockID=0 which consists of ThreadID's from 0 to 7

The Zig-Zig operation on this BlockID is that the thread 0,1,3,5 are active, 6 is inactive, 7 is squashed and 2,4 are misspeculated threads.
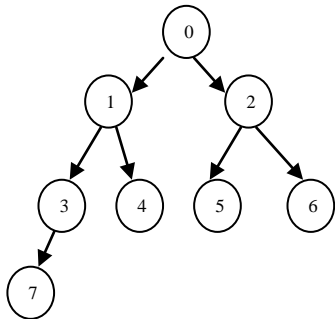

Figure 2. The Binary Search tree of threads of a single Block.

The Figure 3 describes the tree of threads constructed from the 8 threads of the single block.

If t is a node considered for splaying opération.

Parent(t)=(t-1)/2 if t !=0.

Left child (t) =2t+1 if 2t+1 <=n.

Right child=2t+2 if 2t+2 <=n.

Left Sibling (t) = t-1 if t is even.

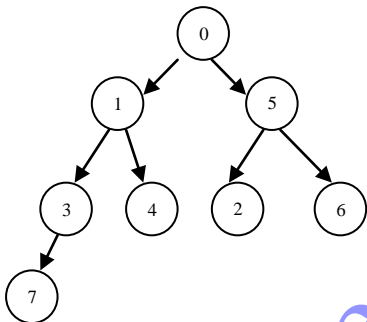Right Sibling (t) =t+1 if t is odd and t+1 <=n [5, 15].

Figure.3 The tree after the splaying operation.

Since it is a self adjusting tree, the nodes which are active are moved near to the root and other nodes are moved to the end.Table1 represent the identification of the parent and child threads on the GPU such that after splaying operation nodes are self adjusted.

Table1. Illustrates the parent and child node relationship.

	0	1	2	3	4	5	6	7	8
Parent p	-	0	0	1	1	2	2	3	3
Left Child LC	1	3	5	7	-	-	-	-	-
Right Child RC	2	4	6	8	-	-	-	-	-
Left Sibling LS	-	-	1	-	3	-	5	-	7
Right Sibling RS	-	2	-	4	-	6	-	8	-

To transfer the data to the speculative threads the table is referred to check for the parent and child threads on the GPU.

At the next level the same operation can be extended to the block level also since the blocks are numbered as the B0, B1, B2 the splaying operation can bring the blocks which are active to the nearer to the root by self adjustment that saves the memory as well as the Collection also.

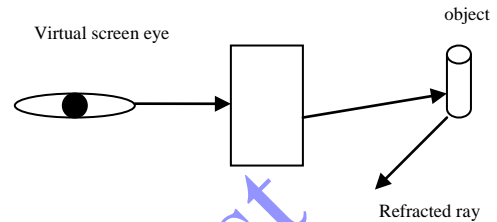## 3. Ray tracing using splaying

Figure 4. Ray Tracing Technique.

In computer graphics, Ray Tracing is a technique for generating an image by tracing the path of the light through pixels in an image plane and simulating the effect of its encounters with virtual objects. In Ray Tracing problem, we send a ray from the eye/camera through each pixel on the virtual screen to compute the color of the pixel. Some light could also have been reflected or refracted by this object. The other light ray is blocked by another object. If the multiple reflections or refractions we trace recursively the reflected or refracted rays until they do not hit any object. Finally the energy contributions of all rays used to get the color of the screen pixel.

Figure 4 describes the ray tracing technique. Ray tracing can be efficiently solved by the technique of speculative parallelization and multithreading. Speculative parallelism technique is a promising technique used in current technologies to enhance the instruction level parallelism and thread level speculation. Speculative multithreading uses speculative Architectural threads where it has two threads. A non speculative thread and a set of speculative threads. A non speculative thread starts executing the program and at the spawning point where the program or loop and instructions can be parallelized. In the Speculative Architectural Thread paradigm, the technique is to execute the different parts of the program in parallel in different sections. The execution of the program results in correct, if the values are computed by the speculative threads and the code executed by them need not be reexecuted by the main thread. Considering the Ray tracing techniques, the execution of the loop or instruction where the ray is being traced or send is traced by the nonspeculative thread. Once the ray is reflected or refracted, the point of reflection or refraction is termed as spawning point or spawning pair [15].

Chen Tian, Rajiv Gupta and Min Feng have developed the Augmented design for the indicating the dynamic data structures with speculative parallelism [4], and also have designed scheduling policies for cross-

iteration dependences as well as irregular control flows [4].

Erik Reinhard and Frederik W. Jansen have discussed about data driven and demand driven tasks for good load balancing and spreading communication evenly in the network [11]. Kobayashi *et al.* [8] proposed hierarchical multiprocessor system for dynamic load balancing with the static one.

Hence from the spawning point or spawning pair, the non speculative thread spawns the speculative thread, the speculative thread starts tracing the ray which is either reflected or refracted. All the rays are traced by the speculative threads and the energy is calculated and passed onto the nonspeculative thread. Each time a ray is refracted or reflected, a speculative thread is spawned and a ray suggests that a nonspeculative thread executes the entire program. T (n) is the time taken for the sequential execution.
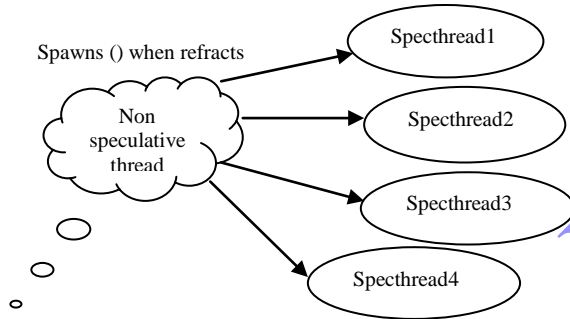

Figure 5. Process of spawning threads.

When speculative threads are tracing the path of a ray, then non speculative thread initializes all the live in variables and initial values to the consumer threads i.e., speculative threads. Figure 5 describes the process of spawning threads. After the proper live in values are propagated to the consumer threads, the consumer threads determines the energy of the rays or the color of the rays. If the value determined by the speculative thread if it is correct, the state of the thread is stored and the values are moved to the producer thread. If the speculation is wrong, the thread is squashed and the nonspeculative thread starts executing normally.

When the speculative thread is spawned by the nonspeculative thread the tracing of the refracted ray is carried out by speculative thread. Then the ray of light passes as it is not hitting the object. The eye/camera is kept at a fixed frame and the ray of the light is moving either in refracted/reflected. i.e., it is in the moving frame. The relativity between the fixed frame and the object moving frame related through the transformations called the LORENTZ TRANSFORMATION [3].

3.1. Lorentz Transformation

It is a linear transformation. This transformation measures the relative distances between the space and the time, the ordering of the events, elapsed time and

preserves the space time interval between the two events [3]. In our paper we apply Lorentz transformation to determine the elapsed time, relativity between the nonspeculative and speculative threads. The time dilation when the speculative threads are spawned.

The eye/camera where the ray of light is spawned is referred as fixed frame. From the fixed frame the co-ordinates are taken as fixed reference values as (x1,y1). Once the ray is passed through the virtual screen, the ray is traced and if it touches any object the ray gets either refracted or reflected. i.e., the ray is in the moving frame. When an observer from the fixed frame is observing the tracing of the ray from a fixed frame, the moving frame, the time dilation happens. The coordinate of the moving frame are treated as (x2,y2). $\Delta t1$ is the time taken for the ray to move from fixed frame. Using this logic we are determining the spawning time of the thread and the thread distance between the speculative thread and the non speculative thread.

When the light ray touches the object the ray gets refracted, the light ray is in the moving frame. The moving frame is moving with velocity v in the x and y directions with respect to the reference frame (fixed frame).

X2=X1-vt/√ 1-v²/c²

$$Y2 = Y1 - vt/\sqrt{1 - v^2/c^2} \qquad (1)$$

To locate the ray from the fixed frame the reverse transformation happens

X1=X2+vt/√1- v²/c²

$$Y1 = Y2 + vt/\sqrt{1 - v^2/c^2} \qquad (2)$$

The time determined as

$$t' = t - vx/c2/\sqrt{1 - v2/c2} \qquad (3)$$

We determine the values of ß and α
Where   α=v/c   ß=1/√1- v²/c²
The distance conserved under a coordinate rotation is
x'=xcosθ+ysinθ   y'=-xsinθ+ycosθ
x'²+y'² =(xcosθ+ysinθ)²+(-xsinθ+ycosθ)²
x'²+y'² =x² cos2θ+2xycosθsinθ+y²sin2θ+x² sin2θ
+2xycosθsinθ+y² cos2θ.
x'²+y'² = x²(cos2θ+sin2θ)+y²(sin2θ+cos2θ)

$$x'2 + y'2 = x2 + y2 \qquad (4)$$

If the ray of light is refracted/reflected in the reference frame the ray gets reflects or refracted.
The coordinate values are (t1,x1,y1) (t2,x2,y2).
The speed of the ray of light in the reference frame of the system are determined as
X'=x2-x1/t2-t1=Δx/Δt

$$Y' = y2 - y1/t2 - t1 = \Delta y/\Delta t \qquad (5)$$

The relativity between the different speeds are
Ux= Δx/Δt

$$= ((\Delta x'+v\Delta t')/\sqrt{1- v^2/c^2})/((\Delta t'+v\Delta x'/c^2)/\sqrt{1- v^2/c^2})$$

$$=\Delta x'/\Delta t' + v/1 + v\Delta x'/\Delta t'c2 \qquad (6)$$

Speculative Time Dilation: The time gap between the spawning of the speculative thread and the spawning pair if it happens in the same coordinate system is referred as correct time of spawning. The correct time is determined as

$\Delta t = t2 - t1$

$=(t2-vx2/c^2/\sqrt{1- v^2/c^2}) - (t1-vx1/c2/\sqrt{1- v^2/c^2})$

$=t2-vx2/c^2 - t1+vx1/ c^2/\sqrt{1- v^2/c^2}$

$=t2-t1/\sqrt{1- v^2/c^2}$

$$=\Delta t/\sqrt{1- v2/c2} \qquad (7)$$

Speculative Distance Contraction: The distance between the spawned speculative thread and the nonspeculative thread is determined as the contraction. The correct distance between the spawned speculative thread and nonspeculative thread L=x2-x1

$T' = (x2-vt2/\sqrt{1- v^2/c^2}) - (x1-vt1/\sqrt{1- v^2/c^2})$

$=x2-x1/\sqrt{1- v^2/c^2}$

$$= L/\sqrt{1- v2/c2} \qquad (8)$$

3.2. A Regular Sturm-Liouville's Equation

The Sturm-Liouville explains a finite dimensional vector space. We consider the inner product of two vectors $X=(X1,X2,X3)$ $Y=(Y1,Y2,Y3)$ which inner product is $X.Y=(X,Y)=X1Y1+X2Y2+X3Y3$. The two nonzero vectors X and Y are said Orthogonal if $X.Y=0$. A set of nonzero vectors is said to be orthogonal if any two distinct vectors from this set are orthogonal. The Sturm-Liouville consists of the general class of boundary value problems with sets of solutions that are mutually orthogonal.

A Sturm-Liouville problem is a boundary value problem on a closed finite interval [a,b] of the form

$$[p(x)y']' + [q(x) + r(x)]y = 0, \quad a < x < b \qquad (9)$$

$$c1y(a) + c2y'(a) = 0 \qquad (10)$$

$$d1y(b) + d2y'(b) = 0 \qquad (11)$$

Where c1,c2 and at least d1,d2 are nonzero and is a parameter.

The nonzero solutions of a Sturm-Lioville problem are called the Eigen functions of the problem and the values which consist of the nonzero solutions are referred as Eigen values.

The Eigen values and Eigen functions of the sturm-lioville problem is defined as

$Y' + \lambda y=0$ $y(0)=y(\pi)=0$.

This equation is similar to the equation (8) with $p(x)=1$, $g(x)=0$ $r(x)=1$.

The boundary conditions a=0 ,b=π with c1=d1=1 c2=d2=0.

Considering the three cases for the solution of the sturm-lioville solution is:

- *Case 1*. when $\lambda<0$, so $\lambda=\alpha^2$ where $\alpha > 0$. the equation becomes $y''- \alpha^2 y=0$ the solution is $y=c1 \sinh \alpha x+c2\cosh \alpha x$. When $y=0$,$c2=0$ when $y(\pi)=0$ the equation will be $0=c1\sinh\alpha \pi$ as $\sinh x \ne 0$ so there are no nonzero solution.

- *Case 2*. when $\lambda=0$, the solution of the equation is $y=c1x+c2$, the boundary conditions c1,c2=0 as there exists no nonzero solution.

- *Case 3*. when $\lambda>0$, $\lambda= \alpha^2$ and $\alpha>0$ so the equation becomes $y''+ \alpha^2y=0$. The solution is $y=c1\cos \alpha x+c2 \sin \alpha x$. When $y(0)=0$ then $0=c1\cos0+c2 \sin0$ so $y=c2\sin \alpha x$.

The other boundary condition is c2!=0 then we get $\sin \alpha \pi=0$ it has the eigen values as

$y1=\sin x$, $y2=\sin2x$, $y3=\sin3x$.....

According to the 3 cases, considering the values of λ, when $\lambda<0$ and when $\lambda=0$ there exists no solution but when $\lambda>0$, there exists solution to the problem and eigen values also exists.

We consider the sphere equation that provides quadratic equation as

$X=-b+\sqrt{b^2 -4ac}/2a$ $X=-b-\sqrt{b^2 -4ac}/2a$

it is similar to representing and replacing $b^2 -4ac$ by λ and considering the cases for finding the roots of the equation. According to the sturm-lioville equation, there exists no solution when $\lambda <0$ and $\lambda=0$. because when $\lambda < 0$, $\lambda=0$ the eigen values remains at nonzero solution. So the ray intersects or passes through the object when $\lambda>0$ as the boundary values for the solution exists or the solution itself exists [2].

The ray is represented as an origin and direction i.e., vector. Origin O=[X0,Y0] Direction D=[Xf, Yf].

The ray consists of points $R(K)=O+Xf*K$.

The ray object intersection is determined by the equation $f(O+Xf*K)=0$.

The ray sphere intersections with center and radius is determined as $(X1-A)^2+(Y1-B)^2=R^2$ where A,B,C are the centres of the Sphere, R is radius and X1,Y1 are the points on the sphere.

The parametric equation for ray are $X=X0+Xf*K$ $Y=Y0+Yf*K$ where X0,Y0 is the origin of the ray Xf,Yf are the camera rays direction. To find the intersection, the ray equation into sphere equation is $(X0+Xf*K-A)^2+(Y0+Yf*K-B)^2=R^2$ which is equal to $(Xf+Yf)^2*K^2+[2[Xf*(X1-A)+Yf*(Y1-B)]]*t+[(Xf-A)^2+(Yf-B)^2-R^2]=0$

The quadratic equation is also in the form of $MX^2+NX+C=0$. Where

$M= [(Xf^2+Yf^2)]$

$N= [2[Xf*(X1-A)+Yf*(Y1-B)]]$

$C= [(X1-A)^2+(Y1-B)^2-R^2]$

Considering the sphere equation above b^2-4ac is discriminant. Based on the values of the discriminant,

- *Case 1*. The solution $b^2-4ac <0$ it leads to imaginary value, the ray and sphere do not intersect in real plane.

- *Case 2*. The solution $b^2-4ac=0$ leads to the boundary conditions with nonzero solutions.

- *Case 3*. If $b^2-4ac>0$ exists in real roots where ray and sphere intersect with each other.

The intersection point of a ray with the plane surface is determined as $AX+BY+D=0$.

The ray equation with the origin and the distance d is represented into the plane equation as $A(X0+Xf*K)+B(Y0+Yf*K)+D=0$.

$AX0+AXf*K+BY0+BYf*K+D=0$ [2].

In this paper, we substitute the discriminant with the value λ. considering the 3 cases of the λ from equations the rays are cast.

Since the discriminant itself is replaced with λ, the computing one square root, 2 multiplications and one subtraction is reduced for each and every computation for computing the color of each pixel in the object which tremendously reduces the computation comparatively.

Considering the values of λ is implemented for tracing the rays. The speculative time dilation and contraction is also considered for optimizing the performance.

3.3. Speculative parallel Ray Tracing

A nonspeculative thread is computing the execution of the ray tracing program where a ray is cast from the eye/camera through the scene which is a image file.

If the ray hits an object, the color of the object is assigned and it gets reflected/refracted. These rays are called secondary rays. The secondary rays are traced by speculative threads parallely.

When the secondary rays are traced by speculative threads, the color of each pixel is not computed but background color is defined. Hence when the secondary rays are tracing the rays there is no need to compute the background colour.

In a speculative parallel ray tracer the algorithm is divided among number of threads on GPU. Since the numbers of processsors on GPU are significantly more, can compute fine grained parallelism to compute the color of the each of the pixel on the image or object simultaneously. The data is distributed from the CPU to the GPU threads. To keep track of the timing events the start and end of the spawning of the threads on non speculative threads. CUDAEVENT () to keep track of the spawned threads to calculate time contraction between start and end of the event.
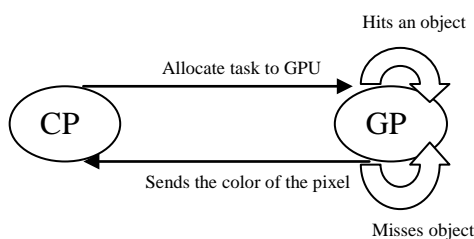

Figure 6.

The steps are as follows. The primary rays shot by the camera or the viewport are stored in the buffered array of the CUDA program and a tree of splay trees are formed and splaying operation is carried out when the rays which are in the moving frame of the rays which hits an object are determined and after splaying operation the rays are traced by the threads on GPU. The rays which hits an object may either gets reflected or refracted such that the speculative threads spawn secondary rays are traced by them. Each time splaying operation is carried for the rays that hit an object. Each of the ray traces the path of the rays and determines the color of the pixels.

For defining the image, OPENGL programming and for ray tracing program, the CUDA is being used. The program is run on GPUOCELOT, a parallel programming simulator for CUDA programming is used for the results and analysis.

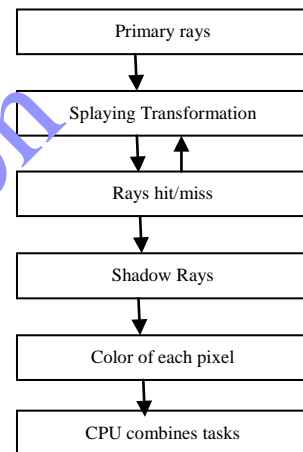The steps for the speculated parallel ray tracer are provided below.

Figure 7.

During the execution of the ray tracing program, the load balancing of the threads is carried out through the technique of splaying operation on GPU's which is discussed in the section 1.1 of this paper. The algorithm of the speculative parallel ray tracer is provided.

Algorithm 3: The algorithm for the speculative parallel ray tracer.

Step1. The image file is created through OPENGL
Step 2. For each pixel of the object
{ color =0;
For (row=0;row< nrows;row+=blocksize
For (col=0;col<ncols; col+=blocksize
{
Step 3. Construct the primary rays.
Step 4. Call splay transform function.
Step5. The nonspeculative thread spawns speculative threads.
Step 6. The rays intersects the object called object intersection.
Step 7. The rays from the camera either hit or miss the objects.
Step 8. The CUDAEVENT records the speculative time dilation and contraction.
Step 9. If the rays hit an object, secondary rays are formed and they either reflect or refract.

Step 10. The GPU processors collect the color of each pixel and group them, finally the CPU determines the colors of the pixels.
}

Algorithm 4: Code for intersection using strum-liouville Equation.

Step 1. Scene is designed using OPENGL with objects, the camera and the light sources are set up.
Step 2. Ray tracer function.
__global__void raytracing (start, direct)
{CUDA_EVENT_T begin, end;
* Float M,N,C,D;*
CUDA_EVENT CREATE (begin)
CUDA_EVENT RECORD (begin,0)
M= Xf^2+Yf^2
N=2(Xf[X1-A]+Yf[y1-B])
C=$(X1-A)^2+(Y1-B)^2-(r*r)$
*D=N*N-4*M*C*
If (D <0)
Return false
Dsqr =sqrt(D)
Dsqr=λ
If (λ>0)
S0=-(N-λ)/2.0
Else
S1=- (N+λ)/2.0
If (λ==0 || λ<0)
Return false
CUDA_EVENT_SYNCHRONIZE (end)
Float executed time
CUDA_EVENT_ELAPSED_TIME (&executed time)

4. Experimental setup

The algorithm is run on the Inspiron5050 using the simulator GPUOCELOT simulator. The efficiency of the algorithm is determined. The amortized complexity of the algorithm is determined as follows.

4.1. Analysis

Amortized Cost: Many of the online algorithm works whose time complexities are determined by the Amortized Time complexities. Amortization is the process of determining the actual cost involved for insertions, deletions and rotation operations. The time complexities remain in O (logn) for the average time complexities. In the worst time analysis remains to be amortized. The average time complexity of the splaying operation follows the Amortized time complexity of O (logn) instead of O(n) and the cost incurred for the insertions and deletions remains the constant or amortized. So this technique is efficiently balances the load without incurring extra time for any primitive operations on the tree. Consider an empty tree and start S splay operations on the tree, the total time of running is O(S logn) but the average time for the operation is O (logn).

The total amortized cost for splaying operation is the sum of all the amortized cost for splay steps.
Amortized cost=∑ cost of splay steps.
$$=\sum 3(ra\ (root)-ra(S))+1$$
$$=\sum 3logn+1$$

=O (logn).
ra(root) is the rank of the root node is logn.
ra(S) rank of the node after splaying.
The performance efficiency of the splay operations on the amortized cost is always O(1) as it is self balancing, self optimized, no storage of data reducing memory requirement and it is very efficient with the uniform accessing of the data.

5. Conclusions

The splay thread cooperation for load balancing results in good performance comparatively on the graphics processors. The amortized time complexities reduce the time of execution task comparatively. The GPU consist of thousands of threads the application of the above technique comparatively reduces the memory access and balancing the work of the active nodes can be realized through the graph.
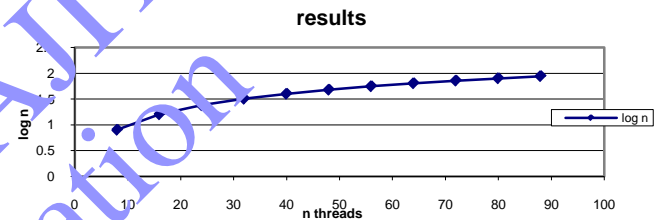
6. Results

Figure 6. graph of nthreads vs logn

The above graph describes as the n number of threads increases for splaying on the GPU's the x axis consists of the n threads and y axis consists of log n resulting in linear speedup.
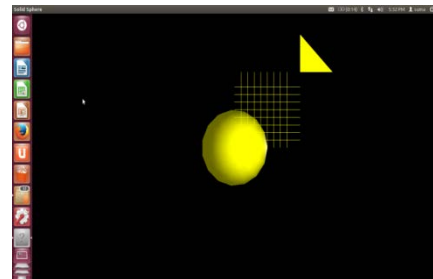

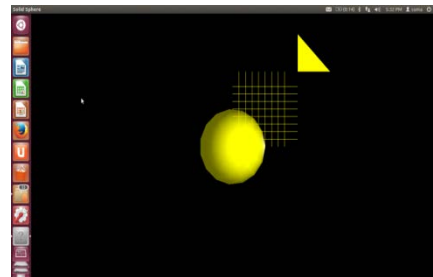Figure 7. Initial scene for Ray tracing.
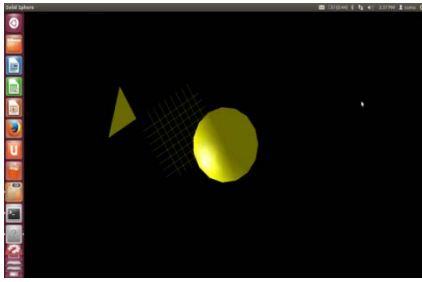

Figure 8. Scene with transformations

Figure 9. Reflections and splaying
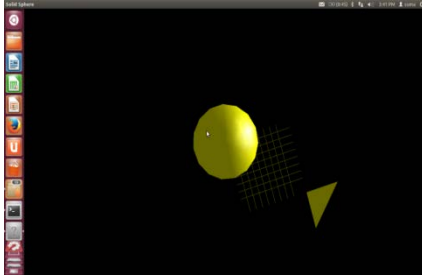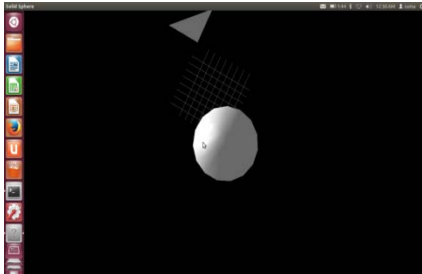

Figure10. Ray tracing image

Figure 11. Ray traced image

Figure 12. Performance on processors

Figure 13. Performance on processors

References

[1] Aguila J L., Campero K.," An Explicit Parallelism Study Based On Thread Level Speculation" *CLEI Electronic Journal*, Volume 17, number2, Paper 8 August 2014.

[2] Asmer N H ., "Partial Differential Equations With Fourier series and Boundary Value Problems", *Pearson, Second Edition*, Chapter 6, pp 333-346.

[3] *CUDA C programming guide* PG-02829001_v5.0 October 2012.

[4] Feng M., Gupta R.,. Bhuyan L N., "Speculative Parallelization on GPGPU's",*PPoPP'12,* February 25–29, 201New Orleans, Louisiana, USA.ACM 978-1-4503-1160.

[5] Horowitz., Sahni, Freed A., "Fundamentals of Data Structures in C", *Computer Science Press*, pp 542-547.

[6] Han M., Wang Z., Yuan J., "Mining Closed and Multi-Supports-Based Sequential Pattern in High Dimensional Dataset", *The International Arch Journal of Information Technology*, Volume 12, No.4, Paper 8, July 2015.

[7] Kaeli D., Yew P C., "Speculative Execution in High Performance Computer Architectures", *Chapman &Hall/CRC* chapter 13, Page no.333-335.

[8] Kobayashi H., Nishimura S., Kubota H.Nakamura T., Shigei,Y.(1988),"Load Balancing Strategies For a Parallel Ray-Tracing System Based On Constant Subdivision", *The Visual Computer4* (4),Pp.197-209.

[9] Lauterback C., Mo Qi, Manocha D., "Work Distribution Methods on GPUs" ,T*echnical Report TR009-16.*

[10] Menon J., Kruijf M., Sankaralingam K., "IGPU: Exception Support and Speculative Execution On GPUs", *39th International Symposium onComputer Architecture, 2012.*

[11] Reinhard E., Jansen F. W.," Rendering Large Scenes Using Parallel Ray Tracing",Volume 23, Issue 7 (1997), *Elsevier Science Publishers* B. V.Amsterdam, The Netherlands.

[12] Sleator., Tarjan, "Self-Adjusting Binary Search Trees", *JACM* Volume 32, No 3, July 1985, pp 652-686.

[13] Tian C, Feng M., Gupta R.,"Supporting Speculative Parallelism in Presence of DynamicData Structures", *PLDI'10* June 5–10, 2010Toronto, Ontario, Canada.

[14] Uht A.K., Morano D., Khalafi A., MDEAlba, Levo K., "A scalable processor with High IPC", *journal of instruction level parallelism,* 5 Aug, 2003.

[15] Wood D., "Data Structures, Algorithms, And Performance",*Addison-Wesley,*1993pp367- 375.

Gopalan Pudur Palaniswamy PhD (IISC Bangalore) M.E. (NIT Trichy) N.P.Gopalan received M.Sc degree with university rank in Mathematics from Madras University in 1978, PhD in Applied Mathematics from Indian Institute of Science, Bangalore in 1983.He received M.E degree in computer Science and Engineering from NIT, Tiruchirappalli and currently serving as the Professor of Computer Applications in it. He has authored books on Web Technology, TCP/IP, Unix Programming, Data Mining object oriented Programming and Signals and Systems. His areas of interest includes Algorithms, Combinatorics, Data Mining, and Distributed Computing

Suma Shivaraju M.C.A., M.Phil. Suma S currently pursuing PhD from Bhartiar University Coimbatore. Her area of interest consists of Parallel and Distributed Computing, Data Structures Analysis and Design of algorithms, Automata Theory and Compiler Design.