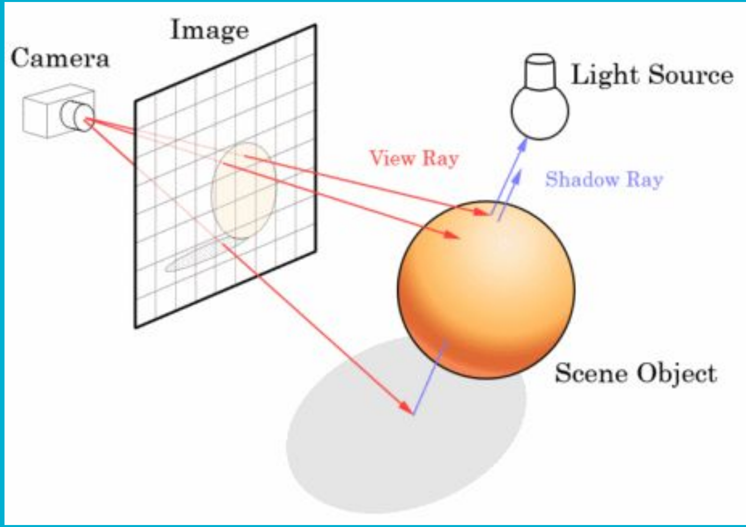
Ray Tracing

Harnessing the power of OpenMP, Pthreads and CUDA

Robert Hughes Sohil Singh

What is Ray Tracing

- Computer Graphic Rendering Technique
- Traces the path of light as pixels in an image plane
- High level of Realism
- High Computational Cost
  - Good for ahead of time rendering
    - Movies, special effects, TV etc.
  - Bad for Real time rendering
    - Video games

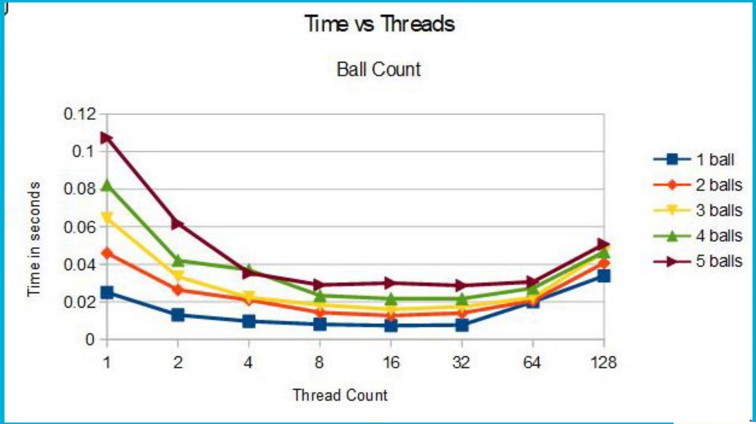


Design of Parallel Solution

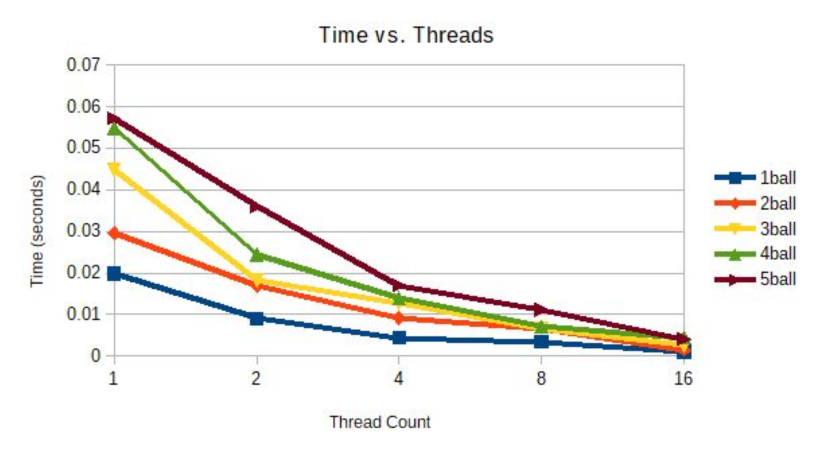
- Pthreads
  - Raw image stored in single array to be written to
  - Each thread accesses a part of the array to write the data to array
  - No critical sections
- OpenMP
  - Similar to Pthreads
- CUDA
  - Use kernels to have every gpu thread map to a single pixel

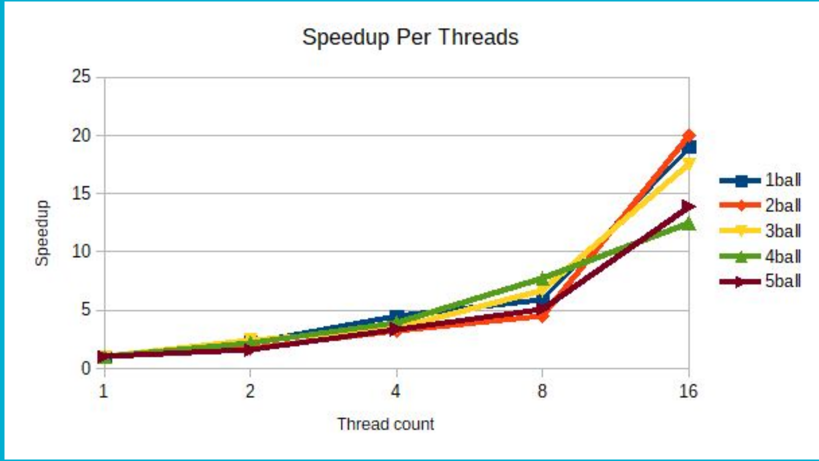
```
printf("I have a rank! %d\n", my_rank);


int x, y;
for(y=my_start;y<my_end;y++){
    for(x=0;x<WIDTH;x++){
```

Results--Pthreads



Results--OpenMP





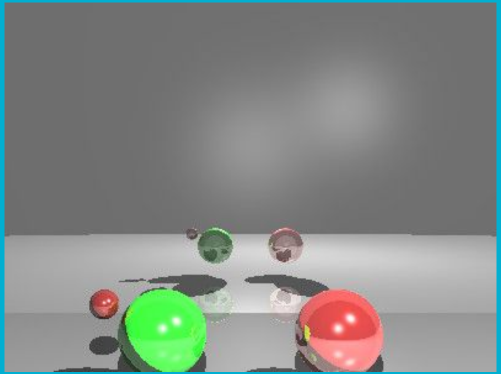
Issues With CUDA

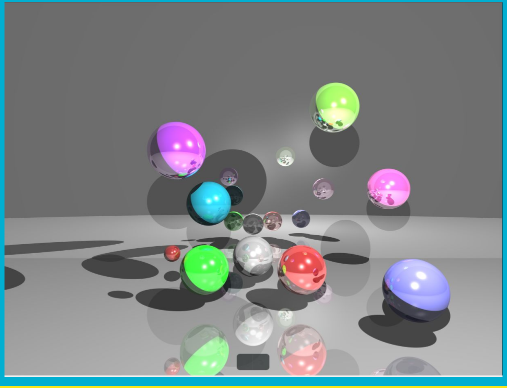
- Too many serial operations
- Global function called host functions
- Fixes:
  - Convert other functions to global
  - Put function in loop

```
do{
    /* Find closest intersection */
    float t = 20000.0f;
    int currentSphere = -1;

    unsigned int i;
    for(i = 0; i < 3; i++){
        if(intersectRaySphere<<<dimGrid, dimBlock>>>(&r, &spheres[i], &t))
                currentSphere = i;
    }
    if(currentSphere == -1) break;

    vector scaled = vectorScale(t, &r.dir);
    vector newStart = vectorAdd(&r.start, &scaled);

    /* Find the normal for this new vector at the point of intersection */
    vector n = vectorSub(&newStart, &spheres[currentSphere].pos);
    float temp = vectorDot(&n, &n);
    if(temp == 0) break;

    temp = 1.0f / sqrtf(temp);
    n = vectorScale(temp, &n);

    /* Find the material to determine the colour */
    material currentMat = materials[spheres[currentSphere].material];

    /* Find the value of the light at this point */
    unsigned int j;
    for(j=0; j < 3; j++){
        light currentLight = lights[j];
        vector dist = vectorSub(&currentLight.pos, &newStart);
        if(vectorDot(&n, &dist) <= 0.0f) continue;
        float t = sqrtf(vectorDot(&dist,&dist));
        if(t <= 0.0f) continue;

        ray lightRay;
        lightRay.start = newStart;
        lightRay.dir = vectorScale((1/t), &dist);

        /* Lambert diffusion */
        float lambert = vectorDot(&lightRay.dir, &n) * coef;
        red += lambert * currentLight.intensity.red * currentMat.diffuse.red;
        green += lambert * currentLight.intensity.green * currentMat.diffuse.green;
        blue += lambert * currentLight.intensity.blue * currentMat.diffuse.blue;
    }
    /* Iterate over the reflection */
    coef *= currentMat.reflection;

    /* The reflected ray start and direction */
    r.start = newStart;
    float reflect = 2.0f * vectorDot(&r.dir, &n);
    vector tmp = vectorScale(reflect, &n);
    r.dir = vectorSub(&r.dir, &tmp);

    level++;

}while((coef > 0.0f) && (level < 15));

img[(x + y*WIDTH)*3 + 0] = (unsigned char)min(red*255.0f, 255.0f);
img[(x + y*WIDTH)*3 + 1] = (unsigned char)min(green*255.0f, 255.0f);
img[(x + y*WIDTH)*3 + 2] = (unsigned char)min(blue*255.0f, 255.0f);
```

Running More Complex Ray Tracer

- Pulled From a Github
- Has more Features
  - Shadows, mirrors, materials
- Ran using https://gpueater.com/console/servers
  - 1CPU Quadro P400 +NVIDIA-410.48
  - 4CPU Quadro P4000 +NVIDIA-410.48
- Ran Tests comparing different Field of View and ball count





Results

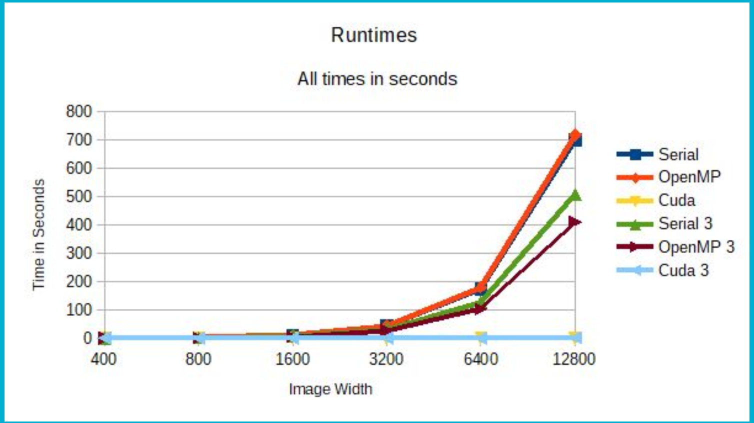
All times in seconds

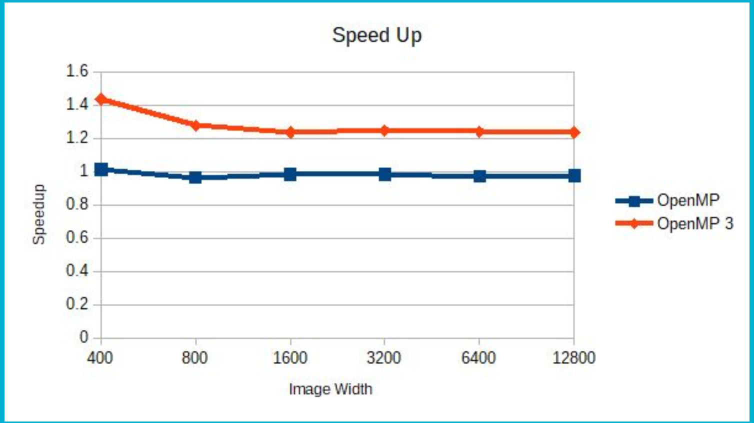
GPUEATER	Quadro P400					
	width	height	FOV	Serial	OpenMP	Cuda
	400	300	35	0.714	0.706	0.354
8 balls	800	600	35	2.732	2.834	0.271
	1600	1200	35	10.983	11.165	0.252
	3200	2400	35	44.155	44.85	0.276
	6400	4800	35	174.699	180.037	0.364
	12800	9600	35	701.353	718.26	0.73
	400	300	60	0.694	0.709	0.268
	800	600	60	2.764	2.827	0.247
	1600	1200	60	10.953	11.237	0.253
	3200	2400	60	45.144	43.782	0.306
	6400	4800	60	174.688	180.625	0.39
	12800	9600	60	703.328	721.05	0.731

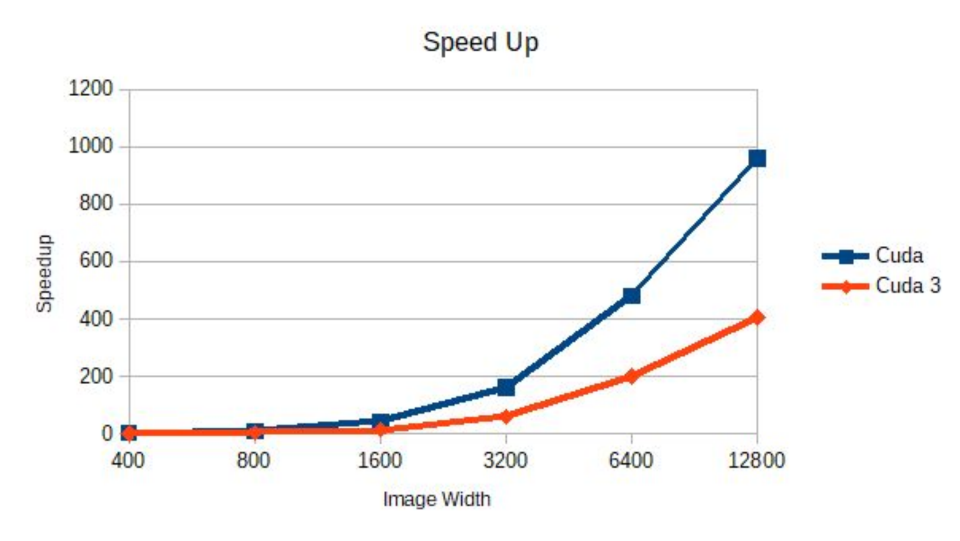
GPUEATER	Quadro P400					
	width	height	FOV	Serial	OpenMP	Cuda
	400	300	35	0.589	0.411	0.49
3 balls	800	600	35	2.061	1.614	0.571
	1600	1200	35	7.944	6.427	0.576
	3200	2400	35	32.004	25.687	0.535
	6400	4800	35	127.427	102.763	0.638
	12800	9600	35	507.657	410.67	1.256
	400	300	60	0.425	0.434	0.569
	800	600	60	1.704	1.691	0.506
	1600	1200	60	6.755	6.671	0.537
	3200	2400	60	54.726	26.844	0.552
	6400	4800	60	107.675	107.418	0.677
	12800	9600	60	430.644	410.617	1.324

GPUEATER	Quadro P1000					
	width	height	FOV	Serial	OpenMP	Cuda
	400	300	35	0.714	0.61	0.354
8 balls	800	600	35	2.732	2.11	0.271
	1600	1200	35	10.983	8.15	0.252
	3200	2400	35	44.155	32.663	0.276
	6400	4800	35	174.699	130.949	0.364
	12800	9600	35	701.353	521.847	0.73
	400	300	60	0.694	0.631	0.268
	800	600	60	2.764	2.211	0.247
	1600	1200	60	10.953	8.605	0.253
	3200	2400	60	45.144	34.371	0.306
	6400	4800	60	174.688	137.589	0.39
	12800	9600	60	703.328	577.58	0.731

GPUEATER	Quadro P1000					
	width	height	FOV	Serial	OpenMP	Cuda
	400	300	35	0.589	0.403	0.49
3 balls	800	600	35	2.061	1.398	0.571
	1600	1200	35	7.944	5.106	0.576
	3200	2400	35	32.004	20.059	0.535
	6400	4800	35	127.427	79.827	0.638
	12800	9600	35	507.657	319.323	1.256
	400	300	60	0.606	0.424	0.569
	800	600	60	2.139	1.437	0.506
	1600	1200	60	8.374	5.386	0.537
	3200	2400	60	33.279	21.449	0.552
	6400	4800	60	131.924	85.039	0.677
	12800	9600	60	530.442	341.739	1.324


Runtimes
All times in seconds
Time in Seconds
800
700
600
500
400
300
200
100
0
400
800
1600
3200
6400
12800
Image Width
Serial
OpenMP
Cuda
Serial 3
OpenMP 3
Cuda 3


Speed Up
Speedup
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
400
800
1600
3200
6400
12800
Image Width
OpenMP
OpenMP 3


Speed Up
Speedup
1200
1000
800
600
400
200
0
400
800
1600
3200
6400
12800
Image Width
Cuda
Cuda 3