GPU Ray Tracing

Leonardo Pereira Medeiros Super Computação

1. Como Rodar o Script

.Instalação das dependências:
sh install.sh

.Compilação do codigo:
make

.Execução do script:
./a.out

.Parametros requisitados ao inicializar o script:
Nx = Largura da imagem
Ny = Altura da Imagem
Ns = Nivel de Anti-Aliasing

2. Introdução

Este projeto consiste em uma adaptação do código de ray tracing em CPU implementado por Peter Shirley e descrito em seu livro Ray Tracing In One Week. Esta adaptação tem como objetivo portar o processamento do algoritmo, que ocorre em CPU, para GPU, com a finalidade de analisar os ganhos de tal alocação, assim como utilizar ferramentas do cuda, API destinada a computação paralela para GPU, para processo de aprendizagem.

3. Ray Tracing

Ray Tracing é um algoritmo de computação gráfica utilizado para síntese (renderização) de imagens tridimensionais, este algoritmo consiste em calcular o trajeto que os raios de luz percorrerem no mundo real. porém partindo da visão do objeto tido como câmera.

No mundo real, os raios de luz são emitidos a partir de uma fonte de luz, percorrendo o espaço até encontrar um objeto. Após os raios de luz atingirem o objeto, estes são refratados ou refletidos, de acordo com as características do objeto, como cor, textura e transparência, alterando assim a sua trajetória, até chegar aos olhos do observador

Dessa forma implementar computacionalmente um sistema que simulasse o que foi descrito no parágrafo anterior, para além de ser incomportável com os meios informáticos que

hoje dispomos, devido à quantidade de processamento exigido, seria um desperdício já que a quantidade de raios de luz que atingem o olho do observador (úteis) é extremamente pequena comparativamente com a quantidade de raios emitidos pela fonte. ,

Assim sendo, o algoritmo ray tracing limita-se a inverter o sentido do trajeto dos raios de luz, passando agora o observador a ser a fonte, e desta forma apenas serão processados os raios que efectivamente são vistos pelo observador, esta técnica é capaz de produzir um grau de foto realismo muito elevado.

4. Processamento em GPU

Enquanto as CPUs possuem um design orientado a minimizar a latência, possuindo poderosas unidades lógicas aritméticas, caches grandes que convertem acessos da memória ram de longa latência para acessos de cache de menor latência, assim como estruturas sofisticadas para controle de latência como branch prediction e data forwarding. As GPUS possuem um design orientado a minimizar o throughput, com unidades aritméticas lógicas simples, obtendo melhor eficiência energética, assim como muitos pipelines para alta taxa de transferência, possui cache reduzidos, que resultam em um acesso a memória contínuo, controles simples de latência e um número massivo de threads para compensar as latências.

Uma forma simples de entender essa diferença é que a CPU tem alguns núcleos otimizados para o processamento sequencial, enquanto uma GPU possui uma arquitetura paralela gigantesca, que consiste em milhares de núcleos melhores criados para lidar com múltiplas tarefas simultaneamente (figura 1).

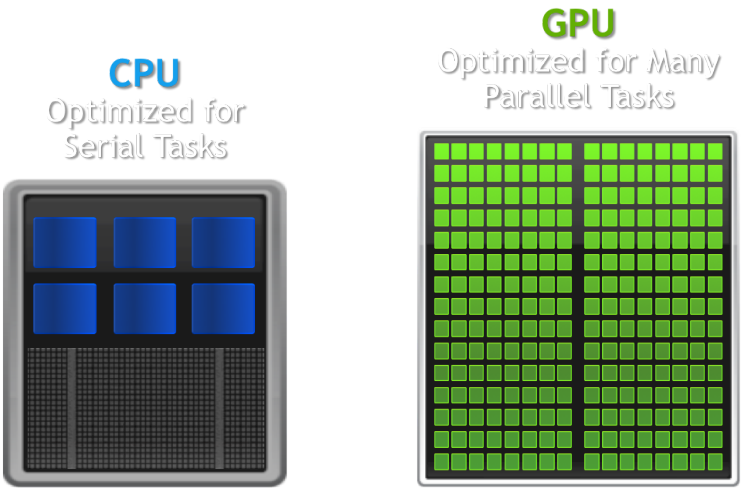

Figura 1. CPU x GPU

Para utilizar todo o poder de processamento da GPU utiliza-se o CUDA(Compute Unified Device Architecture), que é uma engine de computação desenvolvida pela Nvidia, esta

utiliza a estrutura de divisão de memória da GPU em que a thread é a menor unidade de tarefa, essas threads são agrupadas em blocos, em que as threads de um mesmo bloco podem se comunicar e se sincronizar. Os blocos por sua vez são organizados em grids, onde deverão ter a mesma dimensão (figura 2), executando a mesma função de kernel, podendo a grid ser bidimensional ou tridimensional (figura 3).

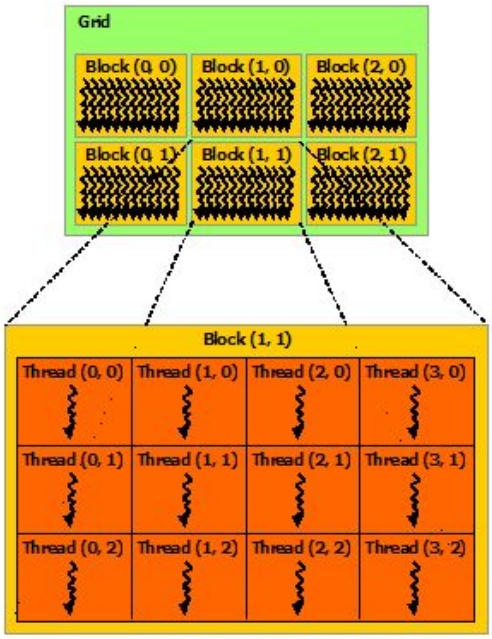

Figura 2. Arquitetura de Memoria GPU

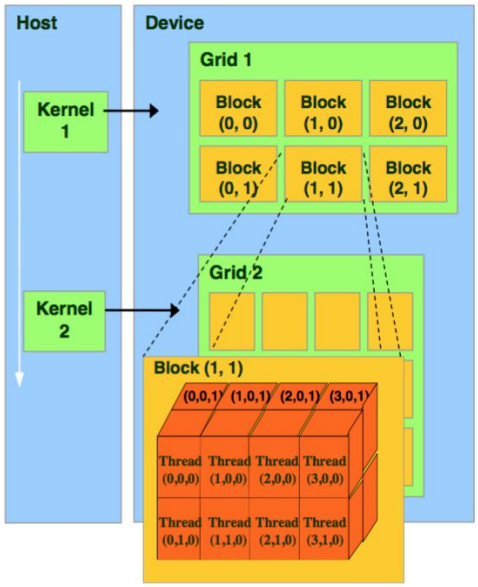

Figura 3. Arquitetura de Memoria GPU

5. Implementação do Codigo

Como dito anteriormente A implementação do código foi feito como adaptação do código de ray tracing em CPU implementado por Peter Shirley, também foi tomado como referência o tutorial do Roger Allen de ray tracing em cuda.

O processo de adaptação foi realizado capitulo por capitulo, tornando todo o processo mais fácil de ser executado, as principais alterações consistem em adicionar flags __host__, __device__ e __global__, que são necessárias na arquitetura do cuda, e indicam nesta ordem que a função será executada em CPU, que a função será executado em GPU, e que a função será um kernel executado em GPU.

Logo foram portados os principais cálculos paralelizáveis do algoritmo para o processamento em GPU, a criação de esferas, o tratamento dos raios, o cenário, o antialising, a aplicação de materiais, todos esses processos de criação e cálculo foram portados.

Como em maior parte os cálculos são realizados para cada pixel da imagem, e estes possuem uma coordenada (x, y), foi criado um bloco bidimensional em que cada thread calcula a cor do pixel com coordenada equivalente.

6. Resultados

Em posse dos dois códigos, com processamento em GPU e com processamento em CPU, dois parâmetros principais foram mensurados, que são o tempo de execução e o consumo de memória para cada script para tamanhos de imagens diferentes, obtendo a tabela 1.

Tamanho da Imagem	GPU Time (seg)	GPU MEM(MiB/11441MiB)	CPU Time (seg)	CPU MEM(%)
300 x 200	1.29688	0	5.12532	0
600 x 400	3.07574	60	20.5131	0
900 x 600	6.62643	92	46.8883	0
1200 x 800	10.6925	96	82.008	0
1800 x 1200	22.0582	110	185.22	0
2400 x 1600	39.6037	128	327.912	0
3000 x 2000	63.3134	154	515.359	0

Tabela 1. Valores de tempo de execução e consumo de memória GPU e CPU.

Embora a memória utilizada pela GPU tenha sido coletada com sucesso, a memória da CPU não apresentou valores com comandos top, rtop, entre outros, tendo sua porcentagem de consumo memoria de 0.0%. Os valores de tamanho de imagem foram escolhidos considerando a diferença de tempo de execução entre cada intervalo assim como seu consumo de memória, a fim de obter um range maior de observação, de forma a analisar diferentes tamanhos de dados.

Pela análise do ganho de tempo de execução portar o código da GPU para a CPU, tem-se como resultado a tabela 2 e a figura 4.

Tamanho da Imagem	GPU Time (seg)	CPU Time (seg)	Ganho
300 x 200	1.29688	5.12532	395.20%
600 x 400	3.07574	20.5131	666.93%

900 x 600	6.62643	46.8883	707.60%
1200 x 800	10.6925	82.008	766.97%
1800 x 1200	22.0582	185.22	839.69%
2400 x 1600	39.6037	327.912	827.98%
3000 x 2000	63.3134	515.359	813.98%

Tabela 2. Valores de tempo de execução GPU e BPU.

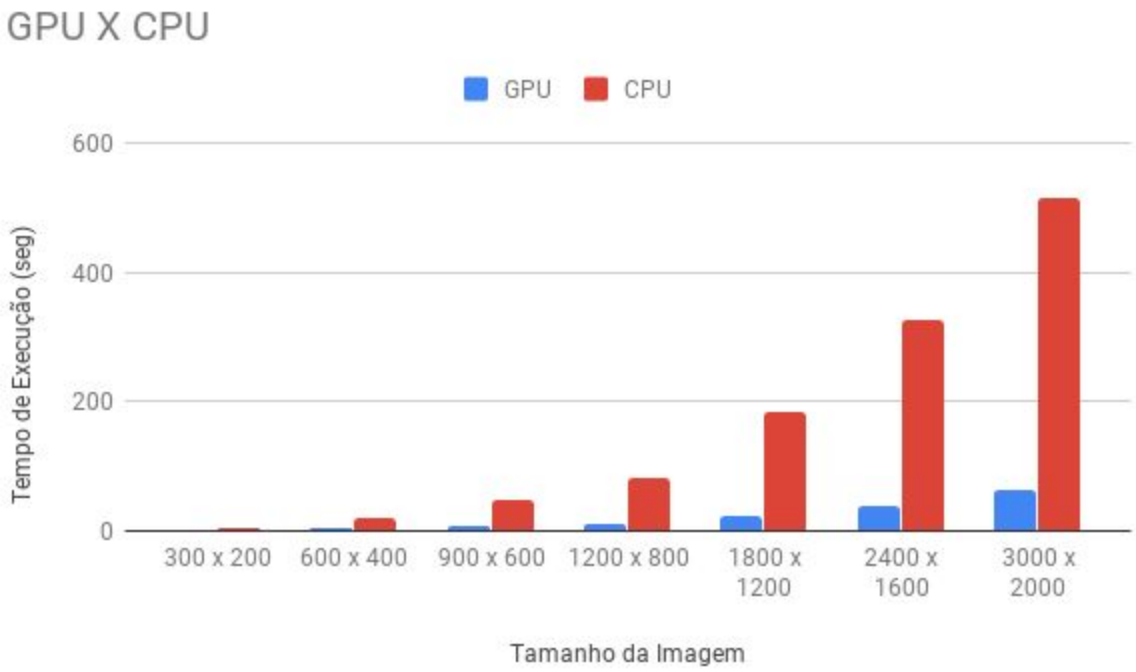

Figura 4. Tempo de execução por tamanho da imagem GPU e CPU.

Como pode-se observar é evidente o ganho de performance quando o algoritmo de ray tracing é configurado para ter suporte em GPU, tendo ganhos ainda mais expressivos quando o tamanho dos dados aumenta.

Maquina utilizada para os testes:

CPU:
Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
Byte Order: Little Endian
CPU(s): 4

On-line CPU(s) list: 0-3
Thread(s) per core: 2
Core(s) per socket: 2
Socket(s): 1
NUMA node(s): 1
Vendor ID: GenuineIntel
CPU family: 6
Model: 79
Model name: Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz
Stepping: 1
CPU MHz: 2704.242
CPU max MHz: 3000.0000
CPU min MHz: 1200.0000
BogoMIPS: 4600.08
Hypervisor vendor: Xen
Virtualization type: full
L1d cache: 32K
L1i cache: 32K
L2 cache: 256K
L3 cache: 46080K
NUMA node0 CPU(s): 0-3

GPU:
GK210GL [Tesla K80]

7. Conclusão

É evidente o ganho de performance quando o algoritmo de ray tracing é configurado para ter suporte em GPU, o que é esperado, por se tratar de um algoritmo com uma estrutura de implementação que permite processamento paralelo.

Como próximos passos pode-se tentar utilizar outros métodos para mensurar de forma mais precisa o consumo de memória de CPU, sendo possível realizar análises gráficas com esse tipo de dado, assim como pode-se testar o ganho de desempenho quando outros parâmetros do algoritmo são alterados, como nível de anti-aliasing, número de esferas, entre outros.

8. Referencias

Peter Shirley:

http://www.realtimerendering.com/raytracing/Ray%20Tracing%20in%20a%20Weekend.pdf

https://github.com/petershirley/raytracinginoneweekend

Roger Allen:
https://devblogs.nvidia.com/accelerated-ray-tracing-cuda/