Paralelização de ray tracing

Paulo Tozzo Ponciano

Introdução

Ray tracing é uma técnica para renderizar imagens tridimensionais, porém ela geralmente ela demora muito tempo para renderizar imagens, para ajudar no seu desempenho um código de ray tracing foi paralelizado usando OpenMP e funções intrínsecas SIMD. O programa parelizado é baseado no programa de ray tracing escrito por Peter Shirley que pode ser encontrado no seguinte github: https://github.com/petershirley/raytracinginoneweekend.

Compilando

O codigo paralelizado esta disponivel em https://github.com/Formulos/raytracinginoneweekend (é um fork do original), compilar ele é fácil, já que ele contém um make, a saída do comando “make” cria um executável chamado “out” que faz a imagem e mostra o tempo que ela demorou para ser gerada, existem também o comando “make npara” que gera um executável “n_out” ele é igual ao “out” mas não é paralelizado, quando o executável for executado ele irá pedir quantos testes serão feitos ou seja quantas vezes o programa rodará, e também qual é o tamanho da imagem desejada.

Paralelização

Primeiramente o código foi alterado para lançar os vetores diretamente em um arquivo,assim não é necessário fazer o redirecionamento da saída do programa, também foi gerada uma imagem de 600 por 400 em vez da original de 1200 por 800 para os testes serem mais rápidos. Logo em seguida foi usado o Intel Vtune rodando na máquina local para identificar quais pontos do código eram mais lentos (ver tabela 1), a máquina usada para testes tem o processador Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz. Com os resultados da tabela foi possível identificar que a região mais problemática que é a função sphere::hit, a função “main” chama essa função dentro de um “for” e este foi paralelizado usando OpemMP for reduction, foi necessário também fazer uma redução customizada já que o OpemMP normalmente não consegue fazer redução de vetores, também foi paralelizado mais dois pontos no código, a função “hitable_list::hit” usando um simples “parallel for” e a geração da imagem usando um “parallel for collapse”.

Function	Module	CPU Time
sphere::hit	out	78.739s
operator-	out	66.920s
dot	out	37.739s
ray::direction	out	35.267s
hitable_list::hit	out	16.094s
[Others]	out	25.759s

Tabela 1 - Vtune hotspots

Testes

Tendo o código paralelizado é possível comparar com ele não paralelizado, os resultados dos testes para uma imagem 600x400 estão na tabela 2 e na imagem 1, o programa não paralelizado usou 4204 kbytes de memória já sua versão paralelizada usou 4600 kbytes. a imagem desejado foi diminuía para 300x200 (tabela 3 e imagem 2) e depois para 150x100 (tabela 4 e imagem 3)

numero do teste	sem parelização	paralelização
1	168.151	108.239
2	170.723	112.389
3	168.257	107.74
4	167.248	107.238
5	162.977	111.377
6	163.566	110.576
7	166.63	103.222
8	167.382	104.196
9	166.698	108.334
10	167.241	106.349

Tabela 2 testes de uma imagem 600x400

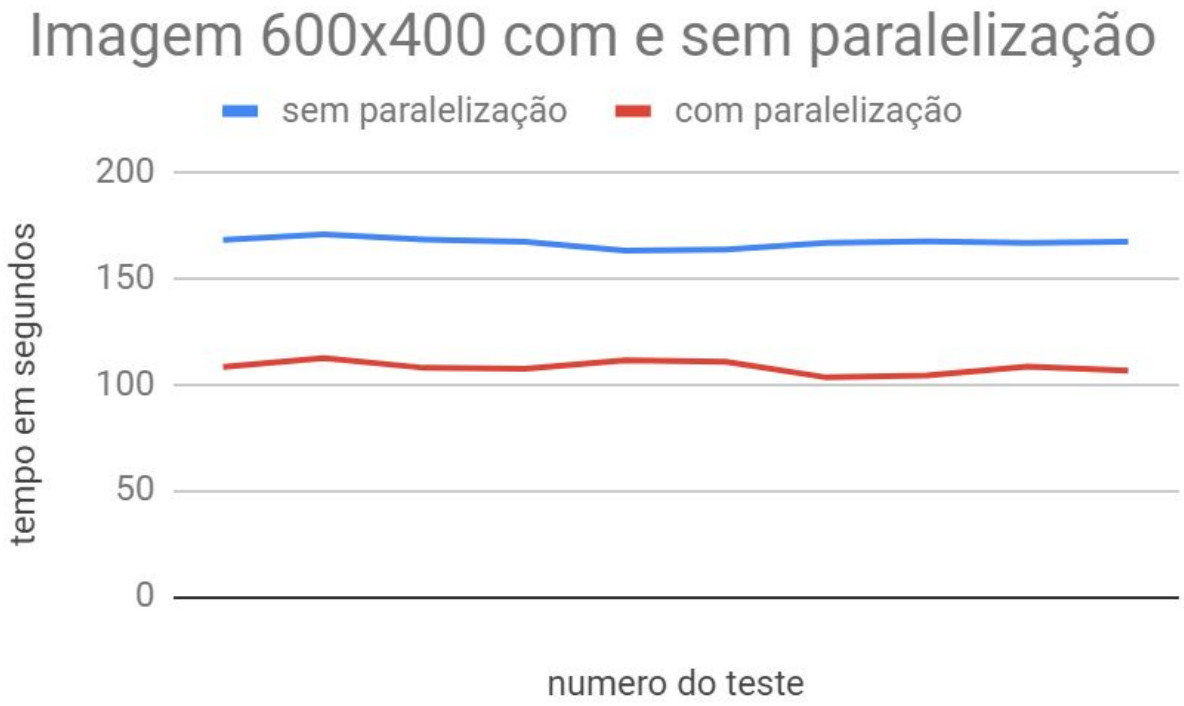

Imagem 1 gráfico dos testes

numero do teste	sem paralelização	com paralelização
1	42.0526	26.6431
2	41.5057	26.842
3	41.7363	25.7544
4	42.2675	27.5645
5	42.3317	27.0867
6	41.2074	25.9233
7	42.2319	25.5497
8	41.5062	25.6868
9	42.5772	26.6091
10	42.331	27.5153

Tabela 3 testes de uma imagem 400x200

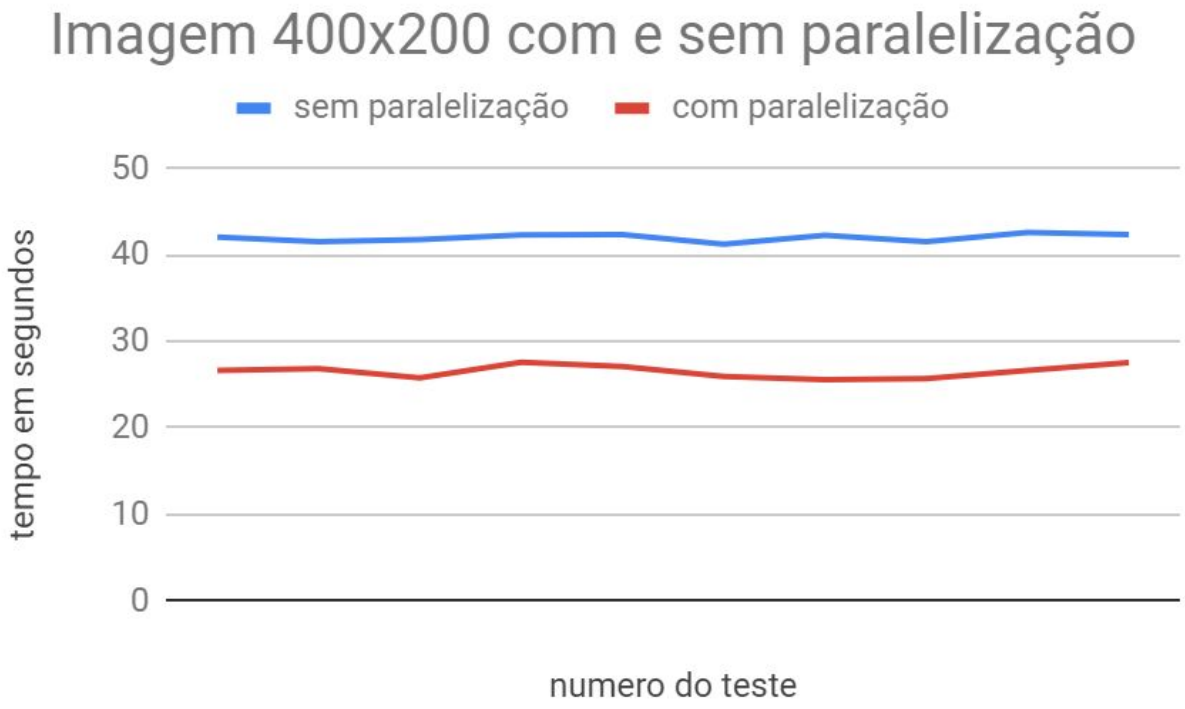

Imagem 2 gráfico dos testes

numero do teste	sem paralelização	com paralelização
1	10.4309	9.13625
2	10.6046	7.83889
3	10.7503	7.23445
4	10.7966	6.81764
5	10.7695	7.4277
6	10.9419	6.97022
7	10.8299	7.11425
8	10.697	7.11417
9	10.8604	7.20615
10	10.9226	6.80246

Tabela 4 testes de uma imagem 150x100

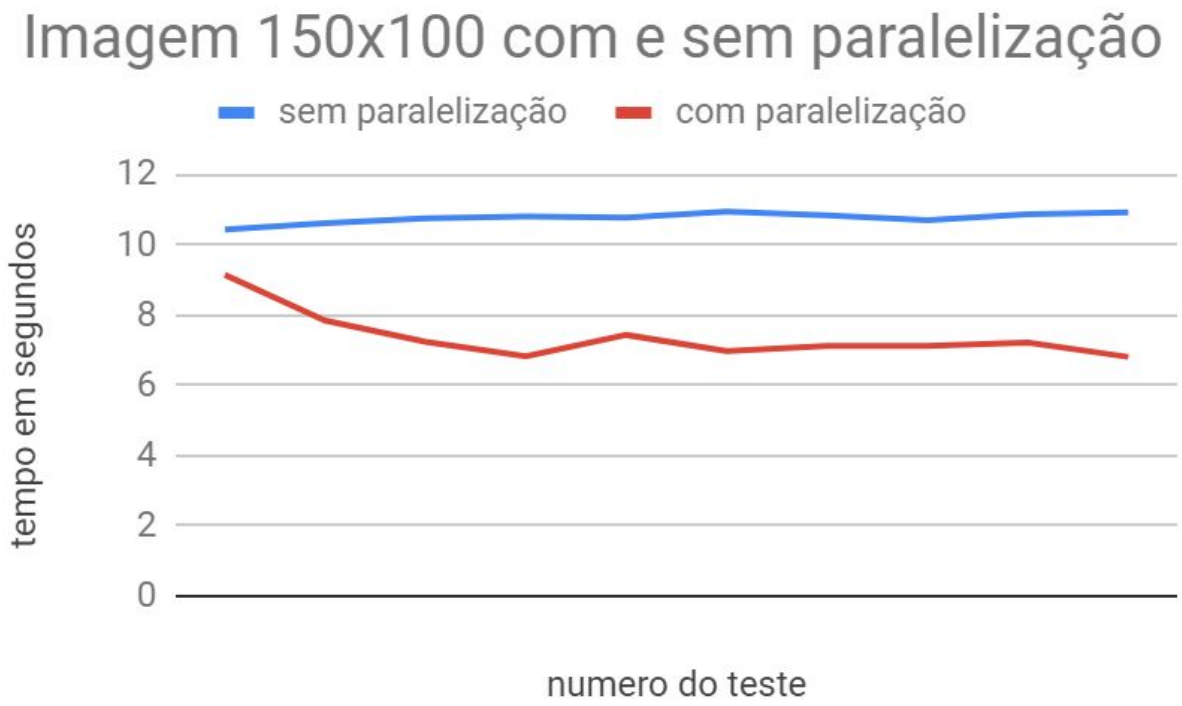

Imagem 3 gráfico dos testes

Conclusão

Podemos ver que o código paralelizado é mais rápido que ele normal, mesmo quando diminuímos o tamanho da imagem ele continua sendo mais rápido, portanto as mudanças na sphere::hit e nas outras funções diminuíram significativamente o tempo em que esse código demora para gerar a imagem usando ray tracing, porém houve um aumento de memória usada entre as duas versões, o que é esperado já que códigos paralelizados precisam guardar mais variáveis.