Benchmark de GPU com simulações N-Body

Alunos:

Rodrigo Vicente Calábria

Gustavo Ebbo Jordão Gonçalves

Ivo Santos Paiva

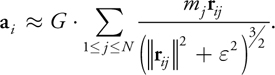
# Introdução

# Baseado no algoritmo apresentado por Lars Nyland Et. Al. em “Fast N-Body Simulation with CUDA”([GPU Gems 3, Capítulo 31), com adaptações para melhor servir para o proposito de benchmarking.](http://developer.nvidia.com/object/gpu-gems-3.html)

O artigo faz o cálculo do benchmark utilizando a simulação de partículas com o algoritmo All-Pairs. A maioria dos benchmarks de GPU focam capacidades gráficas ou capacidades individuais da placa de video. A implementação feita no artigo calcula o benchmark para simulações científicas e a simulacão N-Body foi escolhida, pois representa os requerimentos de muitas simulações. A simulação feita com o algoritmo All-Pairs é de complexidade O(N²), o qual requer padrões de acesso a memória, que são representativos em várias simulações.

## 

Aplicação do algoritmo N-Body

A simulação de partículas envolve o cálculo do movimento de um certo número de partículas (definidas por uma posição, velocidade, massa e possivelmente uma forma).   
As partículas se movimentam de acordo com a lei gravitacional de Newton e se atraem de acordo com a função potencial a seguir:  
  
    
  
Ondeé o fator de “amaciamento”, utilizado para ainda validar a equação acima caso a distância entre os corpos seja nula (evita a divisão por 0).

No código apresentado, o calculo dentro do somatório é realizado pela função “tile\_aceleracao”, e a repetição de somas é realizada por um for dentro da função “calcula\_forca”(que ‘chama’ a função tile\_aceleracao) resultando na aceleração final de um corpo.

Método All-Pairs

O método All-Pairs é o algoritmo mais simples para calcular a força total em cada partícula. A força total em cada partícula (definida pela função potencial de Newton) em relação a outra partícula é calculada uma iteração por vez e as forças finais são utilizadas para calcular a mudança na velocidade e posição da partícula.  
O algoritmo é de complexidade O(N²), pois a aceleração total em cada partícula requer N cálculos.

Código

A função *calcula\_aceleracao* é realizada pela GPU e calcula a aceleração total de um corpo (cada thread calcula a aceleração de um dos corpos).

O vetor de posições é um float4 pois apresenta, além das 3 coordenadas de posição, um valor de massa do corpo.

|  |
| --- |
| \_global\_\_ void calcula\_aceleracao(void \*devX, void \*devA, int ncorpos, int nBlocos){  **//Cria link com os dados da memoria compartilhada**  extern \_\_shared\_\_ float4 posCompartilhada[];  **//Cria vetor com as posicoes na memoria global**  float4 \*globalX = (float4 \*)devX;  **//Cria vetor com as aceleracoes na memoria global**  float4 \*globalA = (float4 \*)devA;  float4 minhaPosicao;  int i, tile;  float3 acc = { 0.0f, 0.0f, 0.0f };  int gtid = blockIdx.x \* blockDim.x + threadIdx.x;  minhaPosicao = globalX[gtid];  **//Faz o calculo em relação a todos os blocos da grid**  for (tile = 0; tile < gridDim.x; tile++) {  int idx = tile \* blockDim.x + threadIdx.x;  **//Carrega as posicoes da memoria global para a compartilhada**  posCompartilhada[threadIdx.x] = globalX[idx];  \_\_syncthreads();  acc = tile\_aceleracao(minhaPosicao, acc);  \_\_syncthreads();  }  **//Salva o resultado na memoria global para o passo da integracao**  float4 acc4 = { acc.x, acc.y, acc.z, 0.0f };  globalA[gtid] = acc4;  } |

A função *tile\_aceleracao* realiza uma das iterações do somatório da lei gravitacional de Newton.

Para referenciar as posições dos outros corpos que estão influenciando no corpo que está sendo processado pela thread, o programa faz uso da memória compartilhada (por blocos, ou *tiles*) para diminuir as transferências da memória global, que é mais lenta.

|  |
| --- |
| \_\_device\_\_ float3 tile\_aceleracao(float4 minhaPosicao, float3 acel){  int i;  **//Cria link com os dados da memoria compartilhada**  extern \_\_shared\_\_ float4 posCompartilhada[];  **//Faz o calculo em relação a todos os corpos do bloco**  for (i = 0; i < TAM\_BLOCO; i++) {  float3 r;  **//calcula a distancia entre os corpos**  r.x = posCompartilhada[i].x - minhaPosicao.x;  r.y = posCompartilhada[i].y - minhaPosicao.y;  r.z = posCompartilhada[i].z - minhaPosicao.z;  **//modulo da distancia ao quadrado + fator de amaciamento**  float distSqr = r.x \* r.x + r.y \* r.y + r.z \* r.z + FATOR;  **//eleva a 6 potencia**  float dist\_a\_6 = distSqr \* distSqr \* distSqr;  **//tira a raiz quadrada e inverte o resultado**  float invDistanciaCubo = 1.0f / sqrtf(dist\_a\_6);  **//multiplica pela massa do corpo**  float s = posCompartilhada[i].w \* invDistanciaCubo;  **//multiplica pelo vetor distancia e realiza mais 1 soma do**  **//somatorio**  acel.x += r.x \* s;  acel.y += r.y \* s;  acel.z += r.z \* s;  }  return acel;  } |

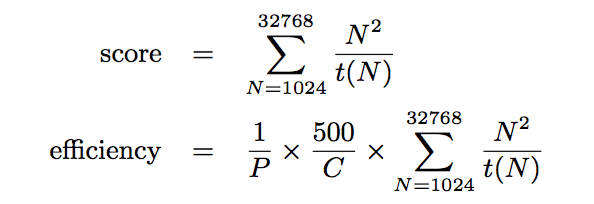
Parte da *main* que chama o processamento em cuda:

|  |
| --- |
| int nBlocks = (nBodies + TAM\_BLOCO - 1) / TAM\_BLOCO;  double totalTime = 0.0;  for (int iter = 1; iter <= nIters; iter++) {  StartTimer();  **//Passa os dados da memoria para a GPU**  cudaMemcpy(d\_buf, buf, bytes, cudaMemcpyHostToDevice);  **//Calcula a aceleracao de cada corpo (cada thread cuida de um corpo)**  **//O terceiro parametro na chamada da funcao em cuda e o tamanho da memoria compartilhada**  calcula\_aceleracao << <nBlocks, TAM\_BLOCO, (TAM\_BLOCO\*sizeof(float4)) >> >(d\_bp, d\_ba, nBodies, nBlocks);  **//Passa os dados da GPU para a memoria**  cudaMemcpy(buf, d\_buf, bytes, cudaMemcpyDeviceToHost);  **//Faz a integracao das posicoes**  for (int i = 0; i < nBodies; i++) {  bp[i].x += ba[i].x\*dt\*dt;  bp[i].y += ba[i].y\*dt\*dt;  bp[i].z += ba[i].z\*dt\*dt;  } |

TAM\_BLOCO = tamanho de threads por bloco.

Resultados

Os resultados do benchmark são obtidos após simular em 1000 iterações em quantidades N de corpos, tais que N = [1024,2048,4096,8192], 500 iterações para N = 16.384 e 250 iterações para N = 32.768. O tempo total realizado para cada tamanho de N é salvo e combinado para a pontuação do benchmark. A simulação irá gerar dois benchmarks, a pontuação e sua eficiência. A pontuação segue a equação(1) e a eficiência a equação(2), onde a variável C é a velocidade de clock e P o número de processadores da GPU.



(1)

(2)

|  |  |  |
| --- | --- | --- |
| Testes | **GTX 970** | **GT 740M** |
| Clock | 1114MHz | 980MHz |
| N° de Cuda Cores | 1664 | 384 |
| Score | 20496760 | 7452884,5 |
| Eficiência | 5528,619579 | 9902,323156 |
| Tempo do Benchmark | 214,352654 s | 590,641522 s |

Assim como obtido no artigo, os testes mostraram que as placas mais atuais, apesar de serem mais rápidas, apresentam menores taxas de eficiência.