PROGRAMAÇÃO EM CUDA

João Victor de Mesquita Cândido dos Santos **RA:** 102028
Raphael Ribeiro Faria **RA:** 104120
Unidade Curricular: Programação Concorrente e Distribuída
Docente: Dr. Álvaro Luiz Fazenda

DESCRIÇÃO DO PROBLEMA:
Converta o programa serial para calc. conducao calor 1D (Arquivo: "fcts.c") para executar em GPU.

OBS: Preocupe-se em converter apenas o trecho responsável pelo cálculo das linhas 41 até 46:

```
for (i=1; i<n; i++) {
  u[i]=u_prev[i]+kappa*dt/(dx*dx)*(u_prev[i-1]-2*u_prev[i]+u_prev[i+1]);
  x += dx;
}
u[0] = u[n] = 0.; /* forca condicao de contorno */
tmp = u_prev; u_prev = u; u = tmp; /* troca entre ponteiros */
```

Faça testes com diferentes configurações de grade e blocos variando a quantidade de threads por bloco da seguinte forma: 512, 256, 128, 64. Verifique em que situação o algoritmo melhorou o desempenho.

SOLUÇÃO:

Para paralelizar o código, foi feito uma função add, conforme pode ser observado abaixo.

```
__global__ void add(double *u, double *u_prev){
  int index = threadIdx.x + blockIdx.x * blockDim.x;
  if(index < N)
    u[index] = u_prev[index] + kappa*dt/(dx*dx)*(u_prev[index - 1] - 2 * u_prev[index] + u_prev[index
+ 1]);
}
```

Como as variáveis se mantiveram iguais ao código base, o n, antes calculado na main, foi calculado fora e disposto como uma variável global. Essa decisão foi tomada, pois os valores que calculam n se mantém constante durante todo o código.

$$N = \frac{tam}{dx} = \frac{1.0}{0.00001}$$

Para garantir a execução correta do código em Cuda, as diretivas cudaMemcpy foram colocadas dentro do while que chama o add, conforme trecho abaixo. Se essas diretivas forem colocadas fora do while, o código dará segmentation fault. Isso, pois ao executarmos uma ação em um kernel as cópias entre host - device e device - host são necessárias para garantir a correta execução do código.

```
while (t<T) {
    x = dx;
    cudaMemcpy(a, u, size, cudaMemcpyHostToDevice);
    cudaMemcpy(b, u_prev, size, cudaMemcpyHostToDevice);
```

```
  add<<<blocks,MAX_THREAD>>>(a, b);
  cudaMemcpy(u, a, size, cudaMemcpyDeviceToHost);
  u[0] = u[N] = 0.; /* forca condicao de contorno */
  tmp = u_prev; u_prev = u; u = tmp; /* troca entre ponteiros */
  t += dt;
 }
```

O tempo foi medido utilizando cudaEventElapsedTime(&temp mili, start, stop). O resultado já é calculado em milissegundos e designado na variável temp mili. O start e stop indicam onde começa e termina o trecho que deve ser medido por meio de cudaEventRecord(start) e cudaEventRecord(stop).

Todos os resultados obtidos podem ser conferidos nas figuras 2, 3, 4 e 5 em teste no cluster. Ao inserir a medição de tempo no código original, conforme mostrado abaixo, foi obtido a figura 1.

```
gettimeofday(&inicio, NULL);
 t = 0.;
 while (t<T) {
  x = dx;
  for (i=1; i<n; i++) {
   u[i] = u_prev[i] + kappa*dt/(dx*dx)*(u_prev[i-1]-2*u_prev[i]+u_prev[i+1]);
   x += dx;
  }
  u[0] = u[n] = 0.; /* forca condicao de contorno */
  tmp = u_prev; u_prev = u; u = tmp; /* troca entre ponteiros */
  t += dt;
 }

 /* Calculando o maior valor e sua localizacao */
 maxloc = 0;
 for (i=1; i<n+1; i++) {
  if (u[i] > u[maxloc]) maxloc = i;
 }
 gettimeofday(&final, NULL);
 printf("Maior valor u[%ld] = %g\n", maxloc, u[maxloc]);
 tmili = (unsigned int) (1000*(final.tv_sec - inicio.tv_sec) + (final.tv_usec -
inicio.tv_usec)/1000);
 printf("%d\n", tmili);
```

```
Inicio: qtde=100000, dt=1e-06, dx=1e-05, dx²=1e-10, kappa=0.000045, const=0.450000
Iteracoes previstas: 10000
dx=1e-05, x=1, x-dx=0.99999
u_prev[0,1]=0, 0.002
u_prev[n-1,n]=0.004, 0.002
Maior valor u[50000] = 99.8486
20079
```

Figura 1 - Código Serial

Os resultados usando Cuda mostraram uma melhora no tempo quando comparados à solução serial, conforme mostrado pela tabela 1. Dentre as threads testadas, apesar de estarem em um intervalo semelhante, o desempenho de tempo utilizando 512 threads foi o melhor e o pior

ocorreu quando se utilizou poucas threads,64 no caso. O *speedup* foi calculado seguinte a seguinte fórmula

$$Speedup = \frac{Tempo\ Serial}{Tempo\ CUDA}$$

Threads por Bloco	Speed Up
512	3.247
256	3.254
128	3.255
64	3.258

Tabela 1 - Speed Up.

Uma possı´vel explicação advém da questão de sincronização entre os blocos: threads sincronizam muito mais rapidamente se estarem no mesmo bloco de execução, porém residem no mesmo kernel da GPU e assim executam mais lentamente. Portanto, pode-se concluir que 512 poderia ser o número ótimo no embate entre número de blocos para um melhor desempenho. Entretanto analisando as figuras abaixo sobre o tempo de execução se percebe que o tempo de execução seguiu um valor padrão fazendo com que o algoritmo mantenha um mesmo nível de paralelismo.

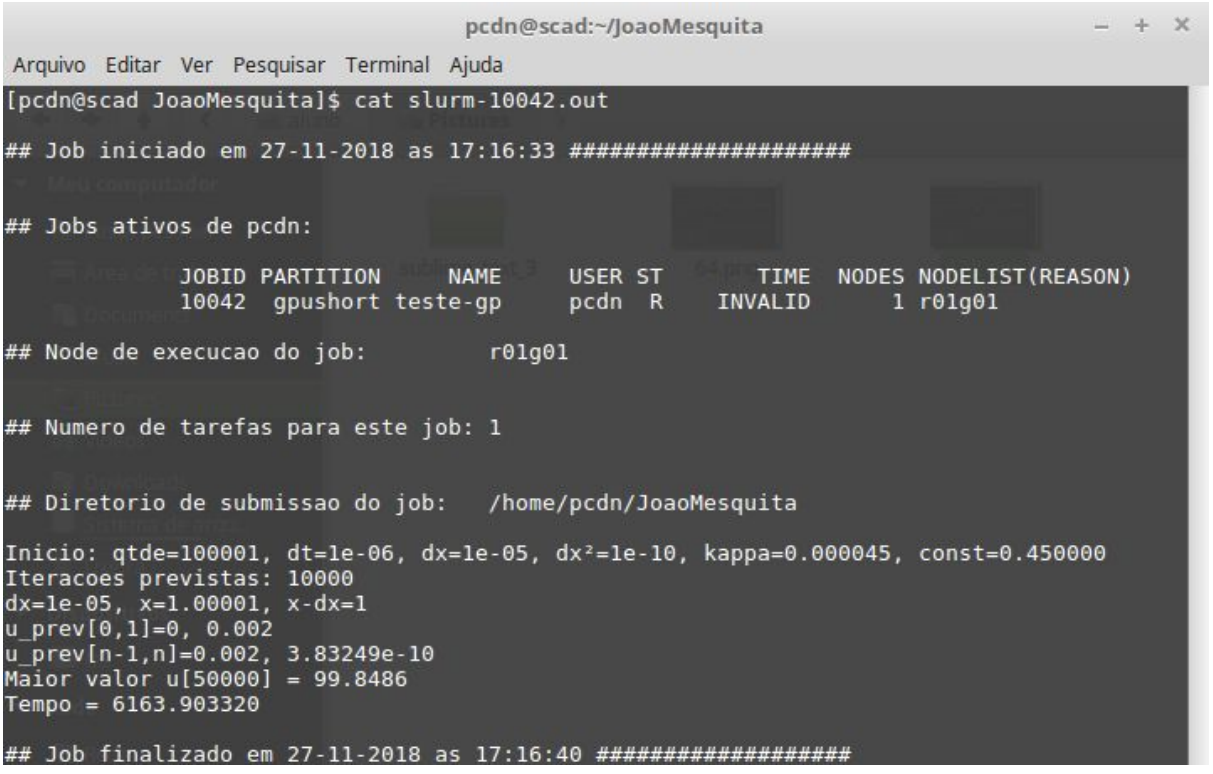

Figura 1. 64 Threads

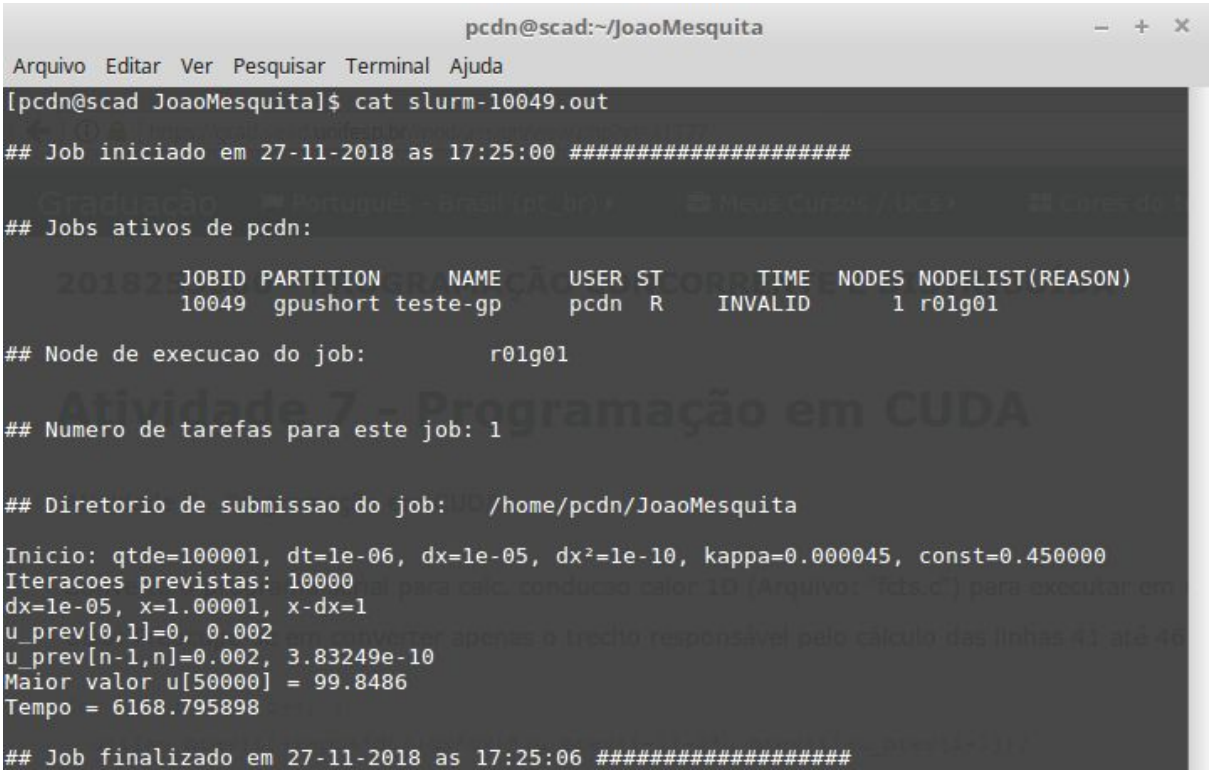

Figura 2 - 128 Threads

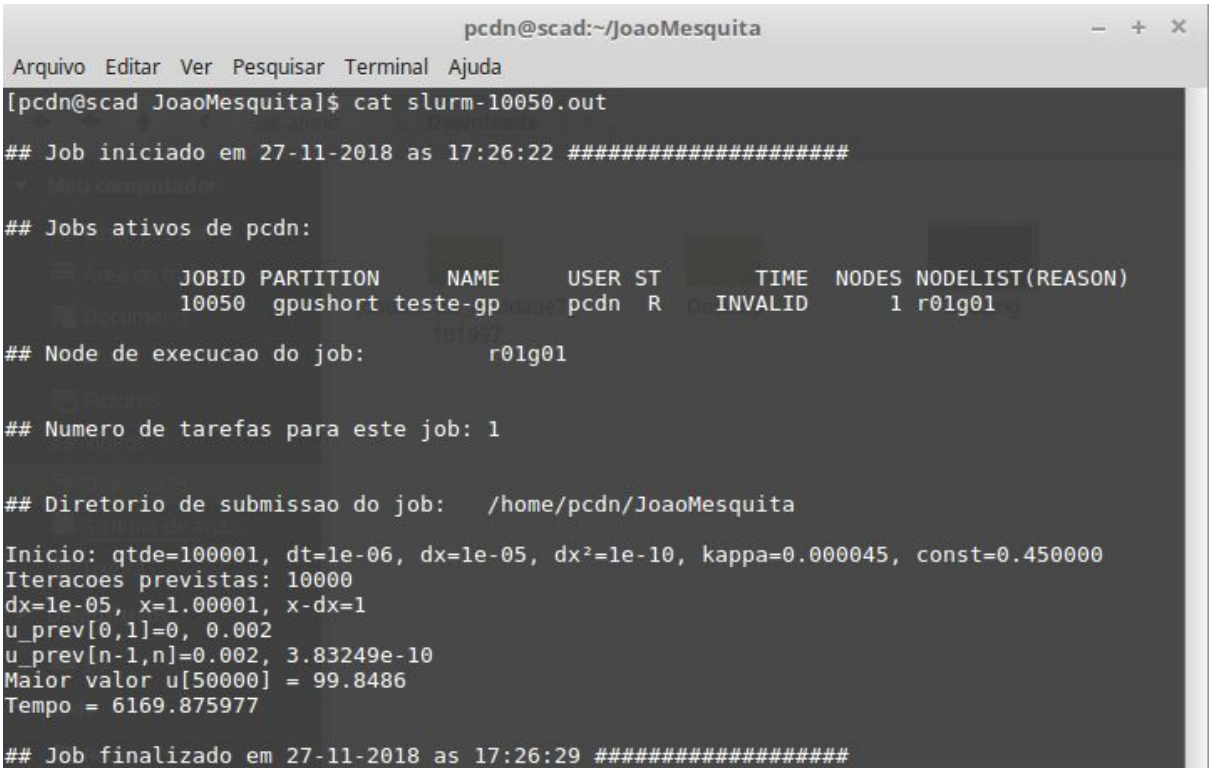

Figura 3 - 256 Threads

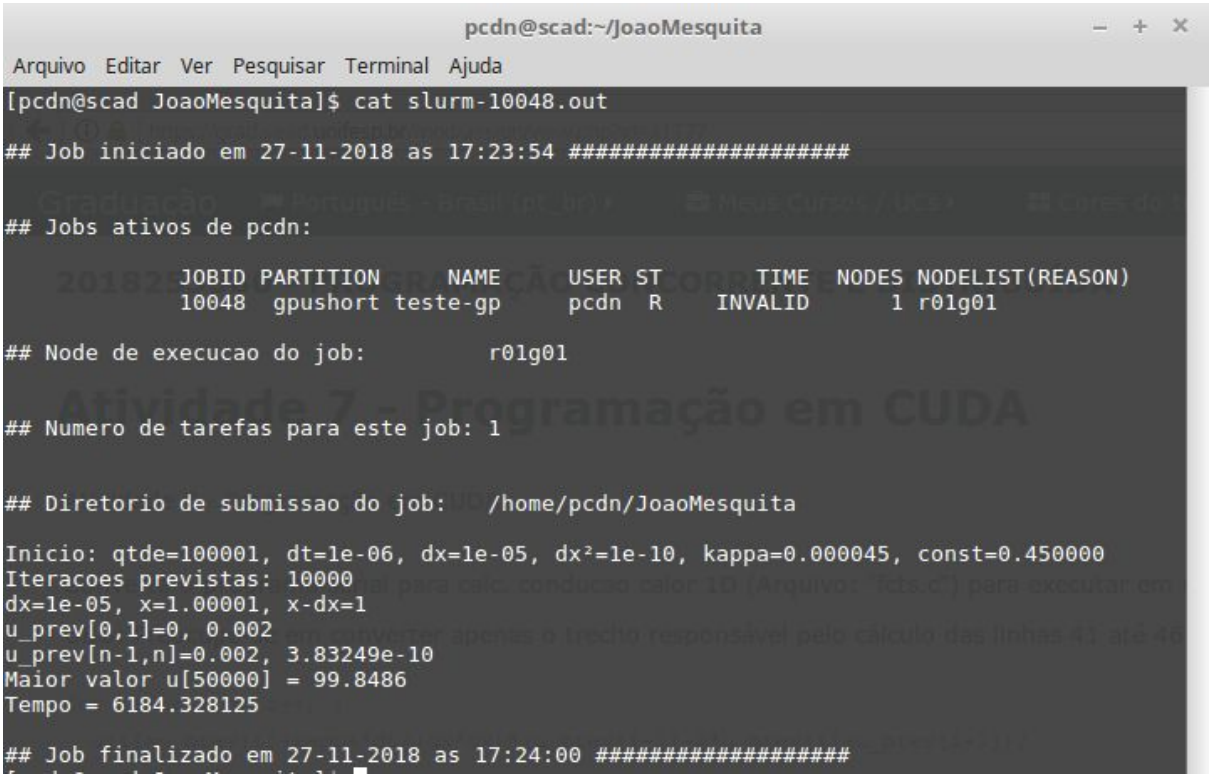

Figura 4 - 512 Threads