Paralelização do Crivo de Erastótenes com OpenMP

Programas ModVida.cu e MainVida.cu modificados

```
// MainVida.cu
  // CPU
  // aloca e inicializa tabuleiros

  t0 = wall_time();
  tabulIn  = (int *) malloc (size);
  tabulOut = (int *) malloc (size);
  InitTabul(tabulIn, tabulOut, tam);

  // dump tabuleiro inicial

  sprintf(msg,"Inicial CPU");
  DumpTabul(tabulIn, tam, 1, 4, msg);

  // avanca geracoes

  t1 = wall_time();
  for (i=0; i<2*(tam-3); i++) {
    UmaVida (tabulIn, tabulOut, tam);
    UmaVida (tabulOut, tabulIn, tam);
  }
  t2 = wall_time();

  // dump tabuleiro final

  sprintf(msg,"Final CPU");
  DumpTabul(tabulIn, tam, tam-3, tam, msg);

  // Correcao na CPU

  if (Correto(tabulIn, tam))
    printf("**RESULTADO CORRETO NA CPU**\n");
  else
    printf("**RESULTADO ERRADO NA CPU**\n");

  t3 = wall_time();

  // Tempos na CPU
```

```
printf("tam=%d; tempos na CPU: init=%f, comp=%f, fim=%f, tot=%f \n",

// GPU
// aloca e inicializa tabuleiros
int* tabulIn_d;
int* tabulOut_d;
cudaMalloc( (void **) &tabulIn_d, size);
cudaMalloc( (void **) &tabulOut_d, size);

t0 = wall_time();
tabulIn  = (int *) malloc (size);
tabulOut = (int *) malloc (size);
dim3 dB (tamBlk, tamBlk);
dim3 dG (nBlk, nBlk);
InitTabul(tabulIn, tabulOut, tam);

// Envia tabuleiro inicializado para a gpu
cudaMemcpy(tabulIn_d, tabulIn, size, cudaMemcpyHostToDevice);
cudaMemcpy(tabulOut_d, tabulOut, size, cudaMemcpyHostToDevice);

// dump tabuleiro inicial

sprintf(msg,"Inicial GPU");
DumpTabul(tabulIn, tam, 1, 4, msg);

// avanca geracoes

t1 = wall_time();
for (i=0; i<2*(tam-3); i++) {
  UmaVidaGpu <<< dG, dB >>> (tabulIn_d, tabulOut_d, tam);
  UmaVidaGpu <<< dG, dB >>> (tabulOut_d, tabulIn_d, tam);
}
t2 = wall_time();

// Pega as variaveis de volta da gpu
cudaMemcpy(tabulIn, tabulIn_d, size, cudaMemcpyDeviceToHost);
cudaMemcpy(tabulOut, tabulOut_d, size, cudaMemcpyDeviceToHost);

// dump tabuleiro final

sprintf(msg,"Final GPU");
DumpTabul(tabulIn, tam, tam-3, tam, msg);

// Correcao na GPU

if (Correto(tabulIn, tam))
```

```
    printf("**RESULTADO CORRETO NA GPU**\n");
  else
    printf("**RESULTADO ERRADO NA GPU**\n");

  t3 = wall_time();

  // Tempos na GPU

  printf("tam=%d; tempos na GPU: init=%f, comp=%f, fim=%f, tot=%f \n",
     tam, t1-t0, t2-t1, t3-t2, t3-t0);


  free(tabulIn);
  free(tabulOut);
  cudaFree(tabulIn_d);
  cudaFree(tabulOut_d);
// ModVida.cu
__global__ void UmaVidaGpu(int* tabulIn, int* tabulOut, int tam) {
  int i, j;
  int vizviv;
  j = threadIdx.x + blockIdx.x * blockDim.x + 1;
  i = threadIdx.y + blockIdx.y * blockDim.y + 1;

  //for (i=1; i<=tam; i++) {
    //for (j= 1; j<=tam; j++) {
      vizviv =
    tabulIn[ind2d(i-1,j-1)] +
    tabulIn[ind2d(i-1,j  )] +
    tabulIn[ind2d(i-1,j+1)] +
    tabulIn[ind2d(i  ,j-1)] +
    tabulIn[ind2d(i  ,j+1)] +
    tabulIn[ind2d(i+1,j-1)] +
    tabulIn[ind2d(i+1,j  )] +
    tabulIn[ind2d(i+1,j+1)];
      if (tabulIn[ind2d(i,j)] && vizviv < 2)
    tabulOut[ind2d(i,j)] = 0;
      else if (tabulIn[ind2d(i,j)] && vizviv > 3)
    tabulOut[ind2d(i,j)] = 0;
      else if (!tabulIn[ind2d(i,j)] && vizviv == 3)
    tabulOut[ind2d(i,j)] = 1;
      else
    tabulOut[ind2d(i,j)] = tabulIn[ind2d(i,j)];
    //}
  //}
}
```

Arquivos de saída da primeira tarefa

```
// Vida_16X04.out
Inicial CPU; Dump posicoes [1:4, 1:4] de tabuleiro 64 x 64
=====
.X..
..X.
XXX.
....
=====
Final CPU; Dump posicoes [61:64, 61:64] de tabuleiro 64 x 64
=====
....
..X.
...X
.XXX
=====
**RESULTADO CORRETO NA CPU**
tam=64; tempos na CPU: init=0.000339, comp=0.003629, fim=0.000005, tot=0.003973
Inicial GPU; Dump posicoes [1:4, 1:4] de tabuleiro 64 x 64
=====
.X..
..X.
XXX.
....
=====
Final GPU; Dump posicoes [61:64, 61:64] de tabuleiro 64 x 64
=====
....
..X.
...X
.XXX
=====
**RESULTADO CORRETO NA GPU**
tam=64; tempos na GPU: init=0.000648, comp=0.002195, fim=0.001294, tot=0.004137
```

```
// Vida_16X08.out
Inicial CPU; Dump posicoes [1:4, 1:4] de tabuleiro 64 x 64
=====
.X..
..X.
XXX.
....
```

```
=====
Final CPU; Dump posicoes [61:64, 61:64] de tabuleiro 64 x 64
=====
....
..X.
...X
.XXX
=====
**RESULTADO CORRETO NA CPU**
tam=64; tempos na CPU: init=0.000339, comp=0.003629, fim=0.000005, tot=0.003973
Inicial GPU; Dump posicoes [1:4, 1:4] de tabuleiro 64 x 64
=====
.X..
..X.
XXX.
....
=====
Final GPU; Dump posicoes [61:64, 61:64] de tabuleiro 64 x 64
=====
....
..X.
...X
.XXX
=====
**RESULTADO CORRETO NA GPU**
tam=64; tempos na GPU: init=0.000648, comp=0.002195, fim=0.001294, tot=0.004137

// Vida_16X16.out
Inicial CPU; Dump posicoes [3:4, 1:4] de tabuleiro 256 x 256
=====
.X..
..X.
XXX.
....
=====
Final CPU; Dump posicoes [253:256, 253:256] de tabuleiro 256 x 256
=====
....
..X.
...X
.XXX
=====
**RESULTADO CORRETO NA CPU**
tam=256; tempos na CPU: init=0.000638, comp=0.239543, fim=0.000023, tot=0.240204
Inicial GPU; Dump posicoes [1:4, 1:4] de tabuleiro 256 x 256
=====
```

```
.X..
..X.
XXX.
....
=====
Final GPU; Dump posicoes [253:256, 253:256] de tabuleiro 256 x 256
=====
....
..X.
...X
.XXX
=====
**RESULTADO CORRETO NA GPU**
tam=256; tempos na GPU: init=0.001125, comp=0.005926, fim=0.053593, tot=0.060644
// Vida_16X32.out
Inicial CPU; Dump posicoes [1:4, 1:4] de tabuleiro 512 x 512
=====
.X..
..X.
XXX.
....
=====
Final CPU; Dump posicoes [509:512, 509:512] de tabuleiro 512 x 512
=====
....
..X.
...X
.XXX
=====
**RESULTADO CORRETO NA CPU**
tam=512; tempos na CPU: init=0.001399, comp=1.915463, fim=0.000073, tot=1.916935
Inicial GPU; Dump posicoes [1:4, 1:4] de tabuleiro 512 x 512
=====
.X..
..X.
XXX.
....
=====
Final GPU; Dump posicoes [509:512, 509:512] de tabuleiro 512 x 512
=====
....
..X.
...X
.XXX
=====
```

```
**RESULTADO CORRETO NA GPU**
tam=512; tempos na GPU: init=0.002766, comp=0.213726, fim=0.215075, tot=0.431567
```

```
// Vida_16X64.out
Inicial CPU; Dump posicoes [1:4, 1:4] de tabuleiro 1024 x 1024
=====
.X..
..X.
XXX.
....
=====
Final CPU; Dump posicoes [1021:1024, 1021:1024] de tabuleiro 1024 x 1024
=====
....
..X.
...X
.XXX
=====
**RESULTADO CORRETO NA CPU**
tam=1024; tempos na CPU: init=0.004587, comp=16.577363, fim=0.000282, tot=16.582232
Inicial GPU; Dump posicoes [1:4, 1:4] de tabuleiro 1024 x 1024
=====
.X..
..X.
XXX.
....
=====
Final GPU; Dump posicoes [1021:1024, 1021:1024] de tabuleiro 1024 x 1024
=====
....
..X.
...X
.XXX
=====
**RESULTADO CORRETO NA GPU**
tam=1024; tempos na GPU: init=0.011201, comp=2.503108, fim=0.835115, tot=3.349424
```

Tempos de execução do laço principal para primeira tarefa

nBlk	Tempo CPU (s)	Tempo GPU (s)
4	0.00354	0.002339
8	0.027702	0.003399
16	0.241555	0.006103
32	1.916376	0.213533

nBlk	Tempo CPU (s)	Tempo GPU (s)
64	16.580344	2.495949

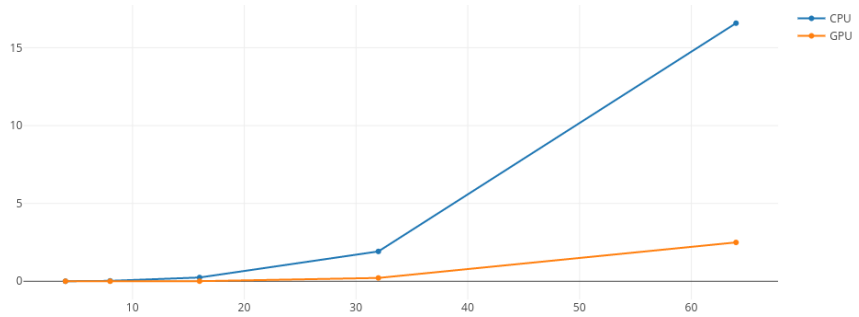

Figure 1: Comparação entre os tempos de execução do loop principal na CPU e na GPU para a primeira tarefa

Análise

Analisando apenas os tempos de execução do loop principal, percebe-se um ápice na diferenca de tempos entre CPU e GPU em nBlk=16, provavelmente por ser um ponto ótimo de acesso à memoria/cache das multiplas threads.

Tempos de execução total da primeira tarefa

nBlk	Tempo CPU (s)	Tempo GPU (s)
4	0.003556	0.004642
8	0.027781	0.011594
16	0.241846	0.060847
32	1.917443	0.431205
64	16.584606	3.338887

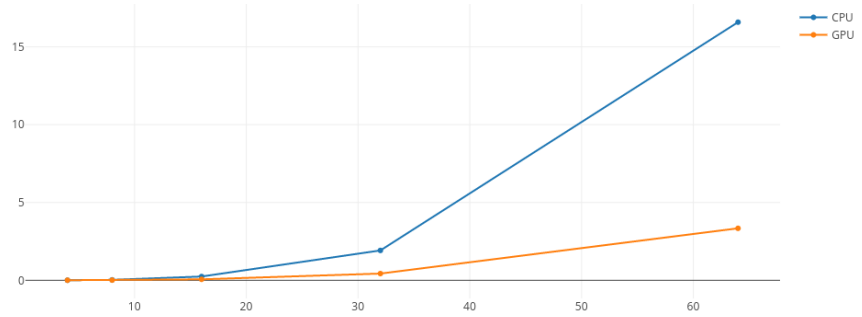

Figure 2: Comparação entre os tempos de execução total na CPU e na GPU para a primeira tarefa

Análise

Apesar de no caso nBlk=16 a GPU ter executado o loop principal 40 vezes mais rapido que a CPU, o tempo total foi apenas 4x maior, uma vez que a transferencia de dados é substancialmente mais devagar que a execução de multiplas computações paralelas. O caso de nBlk=4 mostra que para computações pequenas não vale a pena o tempo de transferencia de dados para a GPU.

Tempos de execução do laço principal para segunda tarefa

nBlk	Tempo CPU (s)	Tempo GPU (s)
4	0.003532	0.002203
8	0.027621	0.003528
16	0.239614	0.006567
32	1.917433	0.046642
64	16.571881	0.504432

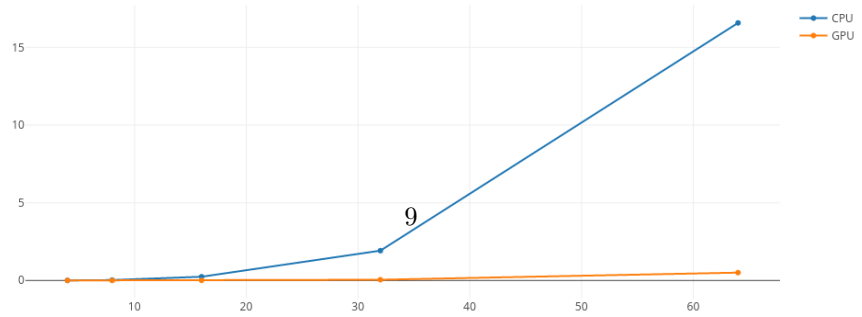

Figure 3: Comparação entre os tempos de execução do loop principal na CPU e na GPU para a segunda tarefa

de i e j de posição muito provavelmente posicinou o tabuleiro na memória de uma forma mais eficiente.

Tempos de execução total da segunda tarefa

nBlk	Tempo CPU (s)	Tempo GPU (s)
4	0.003551	0.003119
8	0.027706	0.005677
16	0.239865	0.017074
32	1.918494	0.096125
64	16.576121	0.683722

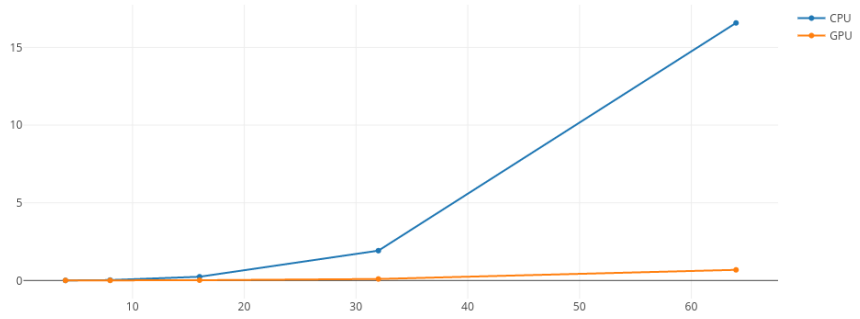

Figure 4: Comparação entre os tempos de execução total na CPU e na GPU para a segunda tarefa

Análise

Os tempos de execução total foram diminuidos basicamente por causa da diminuição do tempo do laço principal, o tempo de enviar os dados para a GPU e pegá-los de volta foi aproximadamente o mesmo.

Conclusões

Com esse experimento pode-se notar o poder de paralelismo que a GPU possui, conseguindo ganhos de velocidade maiores que 10x em alguns casos para o laço

principal. Além disso foi possivel perceber também que a forma como as threads acessam à memória da GPU pode alterar substancialmente o tempo de execução do programa. Foi aprendido também que existe um grande custo envolvido em enviar e pegar os dados da GPU, sendo então uma boa pratica minimizar as trocas de dados e fazer o máximo de contas possível na GPU de cada vez.