Instituto Tecnológico de Aeronáutica

CE-265: Processamento Paralelo

Exercício 06: Executar o Jogo da Vida na GPU

Bruno Benjamim Bertucci - Turma 23.2

1 Introdução

O algoritmo correspondente ao chamado Jogo da Vida foi programado anteriormente nas modalidades sequencial e com paralelismo *OpenMP*. Este relatório detalha uma nova implementação do Jogo da Vida, programada para realizar a computação do avanço de gerações em GPU *NVIDIA* através da plataforma *CUDA*.

As alterações necessárias incluem a adição de novos passos ao programa. Primeiramente, uma nova função foi declarada usando o modificador *__global__*, cujo conteúdo contém o código a ser executado em cada núcleo da GPU. Essa função é denominada *kernel* e, nessa implementação, consiste da atualização de um único elemento do tabuleiro. O código do *kernel* criado está no Anexo A. Além disso, mudanças foram feitas no programa principal do jogo.

Foram adicionados comandos de manipulação de memória da GPU, como a alocação da memória correspondente a uma cópia do tabuleiro, transferência do tabuleiro inicializado para essa memória alocada. Após o término da computação, foram adicionados comandos também para transferir o tabuleiro final da memória da GPU para os espaços correspondentes na CPU, e para liberar a memória alocada na GPU.

No bloco de computação, as chamadas da função *UmaVida* foram substituídas pelo disparo do *kernel* em uma quantidade de blocos da GPU, cada qual inicializando uma dada quantidade de *threads* por bloco, sendo essas quantidades definidas pelo usuário em tempo de execução. A implementação é feita de tal forma que cada *thread* usada em toda a GPU processa um elemento do tabuleiro. Ao final, é feita a sincronização da execução na CPU com a GPU, de tal forma que o programa aguarda o término da computação na GPU antes de prosseguir. O código da função *main* alterada está no Anexo B.

2 Tempos medidos

O programa foi levado ao supercomputador SDumont, onde várias execuções foram efetuadas, todas usando 16 *threads* por bloco, mas variando o número de blocos entre 4, 8, 16, 32, e 64. Além disso, duas variações do *kernel* foram testadas.

Na primeira, as *threads* de uma linha de um bloco recebem elementos do vetor que representa o tabuleiro, espaçados uniformemente entre si, enquanto uma coluna de um bloco recebe elementos consecutivos do vetor. Na segunda variação, ocorre a inversão dessa ordem, ou seja, as *threads* de uma linha recebem posições consecutivas do vetor, enquanto as *threads* de uma coluna recebem posições espaçadas entre si. É esperado que esta tenha um melhor desempenho do que a primeira, devido à coalescência dos endereços da memória possibilitada pela segunda implementação, ou seja, ao agrupamento de endereços consecutivos que resulta em uma quantidade menor de transações necessárias.

2.1 Primeira variação

O teste descrito foi executado usando a primeira variação do *kernel* implementada. Os arquivos de saída estão inclusos no Anexo C. Os tempos da etapa de computação para cada iteração do teste, para a execução na CPU e na GPU, estão na Tabela 1. Já os tempos totais medidos, que

compreendem, além da computação, a inicialização e finalização do programa, estão na Tabela 2.

	4	8	16	32	64
CPU	0.002730	0.021659	0.187208	1.527758	13.226490
GPU	0.003966	0.012612	0.069469	0.504314	3.912821

Tabela 1: Tempos de computação para a primeira variação do teste.

Pode-se observar da tabela acima que a execução na GPU é significativamente mais rápida do que na CPU, com exceção da execução com 4 blocos, que representa o caso de uma quantidade pequena de dados. Além disso, a vantagem da GPU aumenta com o aumento do tamanho da computação.

	4	8	16	32	64
CPU	0.002791	0.021786	0.187581	1.529111	13.230450
GPU	4.758179	5.220618	5.281041	5.704631	9.137961

Tabela 2: Tempos totais para a primeira variação do teste.

Na contramão do que foi observado nos tempos de computação, os tempos totais do programa na GPU são maiores do que na CPU para cargas menores, no caso, até a execução com 32 blocos, de forma que só há uma vantagem observada para a GPU quando usou-se 64 blocos. Além disso, para execuções menores ou iguais a 32 blocos, o tempo total na GPU foi sempre semelhante, apesar das variações da carga computacional. Esses fatos indicam que há um gargalo significativo de alguma etapa além da computação em si. Nesse caso, o gargalo está relacionado às manipulações de memória da GPU, especialmente a alocação do espaço de memória para o tabuleiro.

2.2 Segunda variação

O mesmo teste foi repetido com a segunda variação do *kernel* implementada. Os tempos de computação na CPU e na GPU estão na Tabela 3, enquanto os tempos totais estão na Tabela 4.

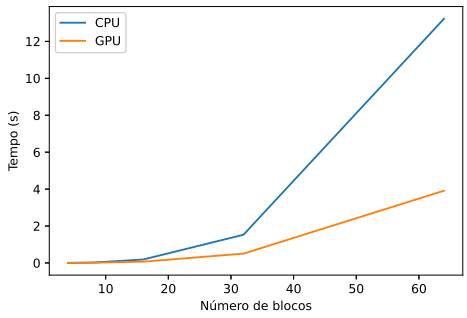
	4	8	16	32	64
CPU	0.002721	0.021659	0.193073	1.526275	13.223412
GPU	0.002375	0.004723	0.017352	0.105827	0.778029

Tabela 3: Tempos de computação para a segunda variação do teste.

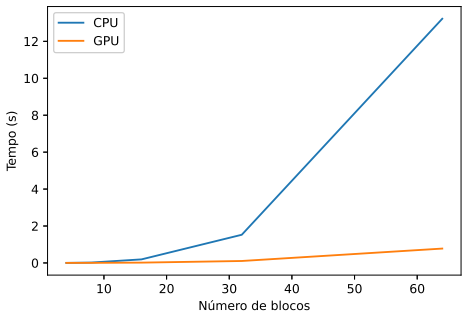
	4	8	16	32	64
CPU	0.002778	0.021813	0.193480	1.527652	13.227668
GPU	5.206189	5.204114	5.219178	5.315783	5.986148

Tabela 4: Tempos totais para a segunda variação do teste.

Os tempos totais registrados com a segunda variação mostram uma tendência semelhante àquela vista com a primeira variação, indicando que o mesmo gargalo age também na segunda implementação. Porém, como os tempos de computação são menores nesse caso, o efeito do gargalo torna-se ainda maior.

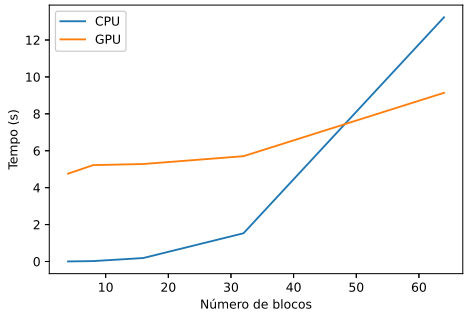


(a) Tempo para a variação 1.

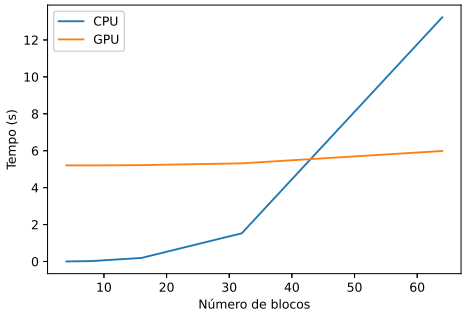


(b) Tempo para a variação 2.

Figura 1: Comparação dos tempos de computação.



(a) Tempo para a variação 1.



(b) Tempo para a variação 2.

Figura 2: Comparação dos tempos totais.

Comparando os tempos de computação na GPU na Tabela 3 com os tempos da na GPU para a primeira variação testada (Tabela 1), observa-se que, conforme esperado, a coalescência permitida por essa segunda implementação resulta em melhores tempos de computação, com uma vantagem que aumenta junto com a quantidade de blocos, alcançando uma execução 5 vezes mais rápida para a execução com 64 blocos. Com essa vantagem, para essa implementação a computação na GPU é mais rápida do que na CPU para todos os testes realizados.

Os tempos de computação medidos nas duas variações podem ser comparados pela Figura 1, e os tempos totais, pela Figura 2.

Anexo A – *Kernel* criado em ModVida.cu

```
__global__ void UmPontoGPU(int* tabulIn_d, int* tabulOut_d, int tam) {
  int i = blockIdx.x*blockDim.x + threadIdx.x + 1;
  int j = blockIdx.y*blockDim.y + threadIdx.y + 1;
  int vizviv;
  vizviv =
    tabulIn_d[ind2d(i-1,j-1)] +
    tabulIn_d[ind2d(i-1,j  )] +
    tabulIn_d[ind2d(i-1,j+1)] +
    tabulIn_d[ind2d(i  ,j-1)] +
    tabulIn_d[ind2d(i  ,j+1)] +
    tabulIn_d[ind2d(i+1,j-1)] +
    tabulIn_d[ind2d(i+1,j  )] +
    tabulIn_d[ind2d(i+1,j+1)];

  if (tabulIn_d[ind2d(i,j)] && vizviv < 2)
        tabulOut_d[ind2d(i,j)] = 0;
  else if (tabulIn_d[ind2d(i,j)] && vizviv > 3)
        tabulOut_d[ind2d(i,j)] = 0;
  else if (!tabulIn_d[ind2d(i,j)] && vizviv == 3)
        tabulOut_d[ind2d(i,j)] = 1;
  else
        tabulOut_d[ind2d(i,j)] = tabulIn_d[ind2d(i,j)];

}
```

Anexo B – Programa MainVida.cu alterado

```
int main(int argc, char *argv[]) {
#define MinTam 4
  int i;
  int tam, tamBlk, nBlk;
  int* tabulIn;
  int* tabulOut;
  size_t size;
  double t0, t1, t2, t3;
  char msg[16];

  // obtem tamanho do tabuleiro

  if (argc != 3) {
    printf("␣uso:␣<exec>␣<celulas␣por␣bloco>␣<quantos␣blocos>\n");
    exit(-1);
  }
  tamBlk = atoi(argv[1]);
  nBlk = atoi(argv[2]);
  tam = nBlk*tamBlk;
  size = (tam+2)*(tam+2)*sizeof(int);

  // tamanho minimo
  if (tam < MinTam) {
    printf("**ERRO**␣tamanho␣%d␣menor␣que␣o␣minimo␣%d\n", tam, MinTam);
    exit(-1);
```

```
}

// execucao na 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",
       tam, t1-t0, t2-t1, t3-t2, t3-t0);

// execucao na GPU
int *tabulIn_d;
int *tabulOut_d;

dim3 dG(nBlk, nBlk);
dim3 dB(tamBlk, tamBlk);

// aloca e inicializa tabuleiros

t0 = wall_time();

InitTabul(tabulIn, tabulOut, tam);
```

```
  cudaMalloc((void **) &tabulIn_d, size);
  cudaMalloc((void **) &tabulOut_d, size);

  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++) {
    UmPontoGPU <<< dG, dB >>> (tabulIn_d, tabulOut_d, tam);
    UmPontoGPU <<< dG, dB >>> (tabulOut_d, tabulIn_d, tam);
  }
  cudaDeviceSynchronize();
  t2 = wall_time();

  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);

  exit(0);
}
```

Anexo C – Arquivos de saída para a primeira variação

16x4

```
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=0.000058, comp=0.002730, fim=0.000003, tot=0.002791
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=4.754162, comp=0.003966, fim=0.000051, tot=4.758179
```

16x8

```
Inicial CPU; Dump posicoes [1:4, 1:4] de tabuleiro 128 x 128
=====
.X..
..X.
XXX.
....
=====
Final CPU; Dump posicoes [125:128, 125:128] de tabuleiro 128 x 128
=====
....
..X.
...X
.XXX
=====
**RESULTADO CORRETO NA CPU**
tam=128; tempos na CPU: init=0.000121, comp=0.021659, fim=0.000006, tot=0.021786
Inicial GPU; Dump posicoes [1:4, 1:4] de tabuleiro 128 x 128
=====
.X..
..X.
XXX.
....
=====
Final GPU; Dump posicoes [125:128, 125:128] de tabuleiro 128 x 128
=====
```

```
....
..X.
...X
.XXX
=====
**RESULTADO CORRETO NA GPU**
tam=128; tempos na GPU: init=5.207910, comp=0.012612, fim=0.000096, tot=5.220618
```

16x16

```
Inicial CPU; Dump posicoes [1:4, 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=0.000353, comp=0.187208, fim=0.000020, tot=0.187581
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=5.211300, comp=0.069469, fim=0.000272, tot=5.281041
```

16x32

```
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=0.001282, comp=1.527758, fim=0.000071, tot=1.529111
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=5.199388, comp=0.504314, fim=0.000929, tot=5.704631
```

16x64

```
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=0.003686, comp=13.226490, fim=0.000274, tot=13.230450
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=5.222181, comp=3.912821, fim=0.002959, tot=9.137961
```