O jogo da vida - Laboratório

Antonio Aldisio 20/2028211
Fernando Miranda 19/0106566
Lorrany Souza 18/0113992

Sem programação paralela ou distribuída

```
tam=8; tempos: init=0.0000319, comp=0.0000761, fim=0.0000520, tot=0.0001600
**RESULTADO CORRETO**
tam=16; tempos: init=0.0000100, comp=0.0006750, fim=0.0000100, tot=0.0006950
**RESULTADO CORRETO**
tam=32; tempos: init=0.0000169, comp=0.0035679, fim=0.0000031, tot=0.0035880
**RESULTADO CORRETO**
tam=64; tempos: init=0.0000172, comp=0.0107958, fim=0.0000100, tot=0.0108230
**RESULTADO CORRETO**
tam=128; tempos: init=0.0000799, comp=0.0844941, fim=0.0000341, tot=0.0846081
**RESULTADO CORRETO**
tam=256; tempos: init=0.0003090, comp=0.6936791, fim=0.0001268, tot=0.6941149
**RESULTADO CORRETO**
tam=512; tempos: init=0.0011911, comp=5.5796621, fim=0.0004869, tot=5.5813401
```

MPI

```
int main(int argc, char** argv) {
  int pow, rank = 0, size = 0;
  int i, tam, *tabulIn, *tabulOut;
  char msg[9];
  double t0, t1, t2, t3;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
```

```
MPI_Barrier(MPI_COMM_WORLD);

MPI_Bcast(tabulIn, (tam+2)*(tam+2), MPI_INT, 0, MPI_COMM_WORLD);

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

t2 = wall_time();

MPI_Gather(rank == 0 ? MPI_IN_PLACE : tabulIn, (tam+2)*(tam+2), MPI_INT,
           tabulIn, (tam+2)*(tam+2), MPI_INT, 0, MPI_COMM_WORLD);

MPI_Barrier(MPI_COMM_WORLD);
```

MPI

```
*RESULTADO CORRETO*
tam=8; tempos: init=0.0000160, comp=0.0002120, fim=0.0001130, tot=0.0003409
*RESULTADO CORRETO*
tam=16; tempos: init=0.0000141, comp=0.0004878, fim=0.0004561, tot=0.0009580
*RESULTADO CORRETO*
tam=32; tempos: init=0.0000460, comp=0.0036259, fim=0.0012212, tot=0.0048931
*RESULTADO CORRETO*
tam=64; tempos: init=0.0000260, comp=0.0104229, fim=0.0098820, tot=0.0203309
*RESULTADO CORRETO*
tam=128; tempos: init=0.0000780, comp=0.0857871, fim=0.0087931, tot=0.0946581
*RESULTADO CORRETO*
tam=256; tempos: init=0.0002990, comp=0.6970198, fim=0.0003932, tot=0.6977119
*RESULTADO CORRETO*
tam=512; tempos: init=0.0011711, comp=5.5661349, fim=0.0055151, tot=5.5728211
```

OMP

```
void UmaVida(int* tabulIn, int* tabulOut, int tam) {
  int i, j, vizviv;

  #pragma omp parallel for private(i, j, vizviv) shared(tabulIn, tabulOut)
  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)];
    } /* fim-for */
  } /* fim-for */
} /* fim-UmaVida */
```

OMP

```
**RESULTADO CORRETO**
tam=8; tempos: init=0.0000069, comp=0.0009410, fim=0.0000479, tot=0.0009959
**RESULTADO CORRETO**
tam=16; tempos: init=0.0000110, comp=0.0006831, fim=0.0000100, tot=0.0007041
**RESULTADO CORRETO**
tam=32; tempos: init=0.0000179, comp=0.0017571, fim=0.0000038, tot=0.0017788
**RESULTADO CORRETO**
tam=64; tempos: init=0.0000160, comp=0.0024400, fim=0.0000110, tot=0.0024669
**RESULTADO CORRETO**
tam=128; tempos: init=0.0000682, comp=0.0162148, fim=0.0000331, tot=0.0163162
**RESULTADO CORRETO**
tam=256; tempos: init=0.0003071, comp=0.1227109, fim=0.0001190, tot=0.1231370
**RESULTADO CORRETO**
tam=512; tempos: init=0.0012040, comp=0.9615560, fim=0.0004630, tot=0.9632230
**RESULTADO CORRETO**
tam=1024; tempos: init=0.0048752, comp=7.7059960, fim=0.0018330, tot=7.7127042
```

CUDA

```
cudaMalloc((void**)&d_tabulIn, (tam+2)*(tam+2)*sizeof(int));
cudaMalloc((void**)&d_tabulOut, (tam+2)*(tam+2)*sizeof(int));

cudaMemcpy(d_tabulIn, tabulIn, (tam+2)*(tam+2)*sizeof(int), cudaMemcpyHostToDevice);

for (i=0; i<2*(tam-3); i++) {
  UmaVida<<<numBlocks, threadsPerBlock>>>(d_tabulIn, d_tabulOut, tam);
  cudaDeviceSynchronize();
  UmaVida<<<numBlocks, threadsPerBlock>>>(d_tabulOut, d_tabulIn, tam);
  cudaDeviceSynchronize();
}

cudaMemcpy(tabulIn, d_tabulIn, (tam+2)*(tam+2)*sizeof(int), cudaMemcpyDeviceToHost);
```

CUDA

```
**RESULTADO CORRETO**
tam=8; tempos: init=0.0000010, comp=0.1340041, fim=0.0000000, tot=0.1340051
**RESULTADO CORRETO**
tam=16; tempos: init=0.0000010, comp=0.0003600, fim=0.0000000, tot=0.0003610
**RESULTADO CORRETO**
tam=32; tempos: init=0.0000081, comp=0.0007350, fim=0.0000000, tot=0.0007432
**RESULTADO CORRETO**
tam=64; tempos: init=0.0000219, comp=0.0014491, fim=0.0000000, tot=0.0014710
**RESULTADO CORRETO**
tam=128; tempos: init=0.0000820, comp=0.0032001, fim=0.0000000, tot=0.0032821
**RESULTADO CORRETO**
tam=256; tempos: init=0.0002751, comp=0.0083330, fim=0.0000000, tot=0.0086081
**RESULTADO CORRETO**
tam=512; tempos: init=0.0009780, comp=0.0338831, fim=0.0000000, tot=0.0348611
**RESULTADO CORRETO**
tam=1024; tempos: init=0.0039670, comp=0.2199140, fim=0.0000000, tot=0.2238810
```

OpenCL

```
// Crie um programa OpenCL
program = clCreateProgramWithSource(context, 1, (const char **)&source, NULL, &err);

// Compile o programa OpenCL
err = clBuildProgram(program, 1, &device_id, NULL, NULL, NULL);

// Crie o kernel OpenCL
kernel = clCreateKernel(program, "UmaVida", &err);

// Criar buffers de memória para tabulIn e tabulOut na memória do dispositivo
bufTabulIn = clCreateBuffer(context, CL_MEM_READ_WRITE,
                            (tam+2)*(tam+2)*sizeof(int), NULL, &err);
bufTabulOut = clCreateBuffer(context, CL_MEM_READ_WRITE,
                             (tam+2)*(tam+2)*sizeof(int), NULL, &err);


// Transferir os dados de entrada para os buffers de memória do dispositivo
err = clEnqueueWriteBuffer(command_queue, bufTabulIn, CL_TRUE, 0,
                           (tam+2)*(tam+2)*sizeof(int), tabulIn, 0, NULL, NULL);
```

OpenCL

```
**RESULTADO CORRETO**
tam=8; tempos: init=0.0000141, comp=0.1743190, fim=0.0000179, tot=0.1743510
**RESULTADO CORRETO**
tam=16; tempos: init=0.0000081, comp=0.1093230, fim=0.0000129, tot=0.1093440
**RESULTADO CORRETO**
tam=32; tempos: init=0.0000131, comp=0.0962379, fim=0.0000122, tot=0.0962632
**RESULTADO CORRETO**
tam=64; tempos: init=0.0000250, comp=0.0968099, fim=0.0000200, tot=0.0968549
**RESULTADO CORRETO**
tam=128; tempos: init=0.0000589, comp=0.0976110, fim=0.0000482, tot=0.0977180
**RESULTADO CORRETO**
tam=256; tempos: init=0.0001981, comp=0.0947869, fim=0.0001452, tot=0.0951302
**RESULTADO CORRETO**
tam=512; tempos: init=0.0007629, comp=0.0985060, fim=0.0005910, tot=0.0998600
**RESULTADO CORRETO**
tam=1024; tempos: init=0.0035279, comp=0.1132841, fim=0.0023069, tot=0.1191189
```

CUDA + OMP

```
cudaMalloc((void**)&d_tabulIn, (tam+2)*(tam+2)*sizeof(int));
cudaMalloc((void**)&d_tabulOut, (tam+2)*(tam+2)*sizeof(int));

cudaMemcpy(d_tabulIn, tabulIn, (tam+2)*(tam+2)*sizeof(int), cudaMemcpyHostToDevice);

for (i=0; i<2*(tam-3); i++) {
  #pragma omp parallel sections
  {
    #pragma omp section
    {
      UmaVida<<<numBlocks, threadsPerBlock>>>(d_tabulIn, d_tabulOut, tam);
      cudaDeviceSynchronize();
    }

    #pragma omp section
    {
      UmaVida<<<numBlocks, threadsPerBlock>>>(d_tabulOut, d_tabulIn, tam);
      cudaDeviceSynchronize();
    }
  }
}

cudaMemcpy(tabulIn, d_tabulIn, (tam+2)*(tam+2)*sizeof(int), cudaMemcpyDeviceToHost);
```

Obrigado.