PARALLEL PROGRAMMING

OPTIMIZING THE HEAT EQUATION ALGORITHM

PROGRAMAÇÃO PARALELA

- A programação paralela é utilizada para realizar atividades que são simples, mas com repetições diversas;
- Processamento dividido para evitar que os núcleos fiquem ociosos.

TECNOLOGIA CUDA

- Plataforma de computação paralela da NVIDIA;
- Permite que o código seja processado nos vários núcleos da GPU;

HEAT EQUATION

- O cálculo consiste entre cada elemento vizinho da matriz (norte, sul, leste e oeste) até que a maior diferença entre o valor antigo e o novo é menor que o Epsilon definido no início da execução do código.

EXEMPLO DE FUNÇÃO UTILIZANDO CUDA

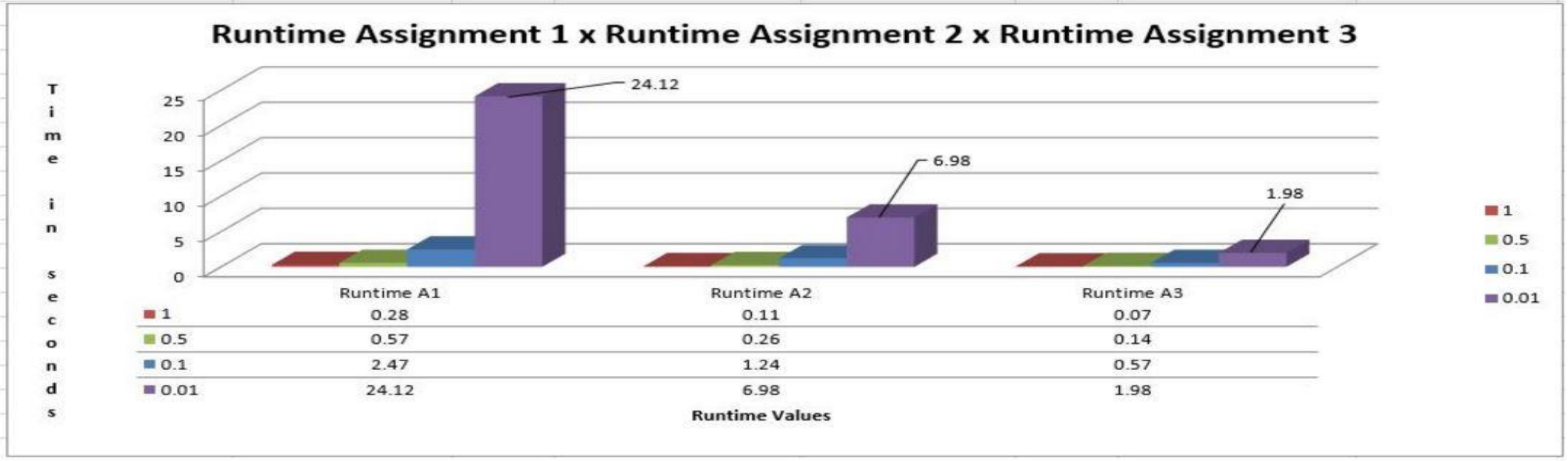
```
__global__ void copyMat(const float *w, float *u){
        int i = blockIdx.x * blockDim.x + threadIdx.x;
        int j = blockIdx.y * blockDim.y + threadIdx.y;
        if (i < M && j < N) {
                u[i * M + j] = w[i * M + j];
        }
        __syncthreads();
}
__global__ void calcHeat(float *w, float *u, float *d, int m, int n, float* d_array){
        int i = blockIdx.x * blockDim.x + threadIdx.x;
        int j = blockIdx.y * blockDim.y + threadIdx.y;
        int tx = threadIdx.x;
        int ty = threadIdx.y;

        __shared__ float s_u[ntpb][ntpb];
        __shared__ float s_w[ntpb][ntpb];
        __shared__ float s_dif[ntpb][ntpb];
        if (tx < ntpb && ty < ntpb) {
                s_w[ty][tx] = w[j * M + i];
                s_u[ty][tx] = w[j * M + i];
        }
        __syncthreads();

        if ( ( tx < (ntpb-1) && ty < (ntpb-1) ) && ( tx >0 && ty > 0 ) && ( i < M && j < N ) ) {
                s_w[ty][tx] = ( s_u[ty - 1][tx] + s_u[ty + 1][tx] + s_u[ty][tx - 1] + s_u[ty][tx + 1] ) / 4.0;

                s_dif[ty][tx] = fabsf(s_w[ty][tx] - s_u[ty][tx]);

                //if (s_dif[ty][tx] < 0){ s_dif[ty][tx] *= -1; }
        }
        __syncthreads();
        if (tx < ntpb && ty < ntpb) {
                w[j * M + i] = s_w[ty][tx];
                //u[j * M + i] = s_w[ty][tx];
                d_array[j * M + i] = s_dif[ty][tx];
        }
        __syncthreads();
}
__global__ void bigDiff(float* d_array, float* d, int m, int n){
        int i = blockIdx.x * blockDim.x + threadIdx.x;

        for (int x = 1; i + x < m*n; x *= 2) {
                if (d_array[i] > *d || d_array[i + x] > *d){
                        if (d_array[i] > d_array[i + x])
                                *d = d_array[i];
                        else
                                *d = d_array[i + x];
                }
                __syncthreads();
        }
}
```

RESULTADOS

A1	
Error Tolerance	Runtime A1
1	0.28
0.5	0.57
0.1	2.47
0.01	24.12

A2	
Error Tolerance	Runtime A2
1	0.11
0.5	0.26
0.1	1.24
0.01	6.98

A3	
Error Tolerance	Runtime A3
1	0.07
0.5	0.14
0.1	0.57
0.01	1.98



MATERIAL DE PROGRAMAÇÃO PARALELA

- https://www.udacity.com/course/intro-to-parallel-programming--cs344
- https://scs.senecac.on.ca/~chris.szalwinski/archives/gpu610.131/pages/content/index.html
- http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-microsoft-windows/#axzz3rEpkLTgD

REFERÊNCIAS

- https://www.udacity.com/course/intro-to-parallel-programming--cs344
- https://scs.senecac.on.ca/~chris.szalwinski/archives/gpu610.131/pages/content/index.html
- http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-microsoft-windows/#axzz3rEpkLTgD
- http://zenit.senecac.on.ca/wiki/index.php/Hu3Team
- Material de aula pertencentes ao Prof. Anderson Moreira.

DÚVIDAS?