ENGENHARIA AEROESPACIAL E MECÂNICA
MECATRÔNICA
CE-265 PROCESSAMENTO PARALELO

Docente: Jairo Panetta
Discente: Lucas Kriesel Sperotto

# Exercício 9 – RELATÓRIO

## Paralelização em CUDA do "Jogo da Vida" no Sistema "CESUP"

## Introdução:

Este relatório esta dividido em quatro partes distintas: procedimento, resultados, conclusões e anexos em arquivo anexado na submissão.

## Procedimento:

Criado pasta e copiado arquivo como solicitado, logo depois, compilado o código fornecido e executado com tamanho de tabuleiro 4X4. O resultado está no arquivo "Vida.out".

Alterado script "Executa.sh" para colocar junto ao nome do arquivo de saída o tamanho do tabuleiro, simplesmente para facilitar a separação dos resultados.

Copiado partes do código como solicitado, e criado "kernel" de execução na GPU. Para a paralelização foi criado tabuleiros para o host e para o device (utilizando sufixos _h e _d), alocado tabuleiro na memória do device (cudaMalloc) e enviado tabuleiro inicializado na CPU para a GPU.

Dimensionado espaço de endereçamento das threads com o uso de variáveis "dim3" (de forma a termos tamBlk e nBlk para x e para y). Depois desta etapa, chamado kernel para iterar as vidas. Entre cada iteração utilizado função de sincronização das threads da GPU (cudaThreadSynchronize()). Terminado laço copiado tabuleiro da GPU para a CPU e verificado correção.

Para a criação do kernel de execução, utilizei os índices i e j como o índice do bloco multiplicado pelo tamanho do bloco mais o índice da thread mais um (respectivamente nas direções x e y). A soma de uma unidade nos índices é para evitar conflito com a zona morta do vetor do tabuleiro. Mais nenhuma alteração foi necessária no código. Creio que separei o código de forma legível e está bem comentado.

## Resultados Paralelização

Na tabela 1 estão mostrados os tempos obtidos para a execução na CPU, e na tabela 2 os tempos para execução na GPU.

**Tabela 1 – Tempos de Execução – CPU**

| Tamanho Tabuleiro | Inicialização | Computação | Finalização | Tempo Total |
|---|---|---|---|---|
| 8 | **0,000039** | **0,000015** | **0,000003** | **0,000057** |
| 64 | **0,000091** | **0,007207** | **0,000022** | **0,007320** |
| 128 | **0,000143** | **0,042952** | **0,000023** | **0,043118** |
| 256 | **0,000548** | **0,388097** | **0,000066** | **0,388711** |
| 512 | **0,002330** | **4,117642** | **0,000374** | **4,120346** |
| 1.024 | **0,008888** | **32,732813** | **0,002443** | **32,744144** |

**Tabela 2 – Tempos de Execução – GPU**

| Tamanho Tabuleiro | Inicialização | Computação | Finalização | Tempo Total |
|---|---|---|---|---|
| 64 | **1,822898** | **0,008454** | **0,000108** | **1,831460** |
| 128 | **1,639179** | **0,021559** | **0,000199** | **1,660937** |
| 256 | **1,901381** | **0,080682** | **0,000597** | **1,982660** |
| 512 | **1,770781** | **0,532513** | **0,002361** | **2,305655** |
| 1.024 | **1,828651** | **3,827751** | **0,007488** | **5,663890** |

Primeiramente vemos que os tempos de inicialização em geral são maiores na GPU, isso se deve ao fato do envio das variáveis da CPU para a GPU. No entanto analisando os tempos de computação vemos um ganho apenas em tabuleiros com mais de 128x128 elementos. O tabuleiro de 64x64 elementos não teve um desempenho na GPU inferior ao da CPU, no tempo de execução, acredito que se deva ao fato do problema não utilizar um numero suficiente de threads do bloco, tendo assim um mau uso da memória local daquele bloco.

Observado que os tempos de finalização são bem inferiores aos tempos de inicialização, infelizmente não medi o tempo da alocação da memória da GPU (individual), Como a função “cudaMemcpy” é síncrona, pode se concluir que o tempo de retorno dos dados ou da GPU é inferior ao tempo de se enviar os dados. Acredito que esta hipótese é incorreta, se analisarmos os tempos de inicialização para cada tamanho de tabuleiro fica claro que a alocação da memória na GPU leva um tempo superior ao envio dos dados (para esses tamanhos de problema.

Agora partindo para o tempo total, temos perda significativa de desempenho para problemas pequenos, pelas perdas na comunicação CPU/GPU e outras variantes do processo, mas em contrapartida, o ganho em problemas maiores é de ordem de 10 vezes. O que nos diz que para problemas que usam significativamente o poder de processamento da GPU, obviamente tiraremos muito desempenho, lembrando que para problemas de granularidade fina.

## Conclusões

Para ilustrar melhor os resultados construí o gráfico 1 que mostra o tempo total de execução da CPUxGPU. Onde podemos ver o momento onde a execução da GPU se torna vantajosa.

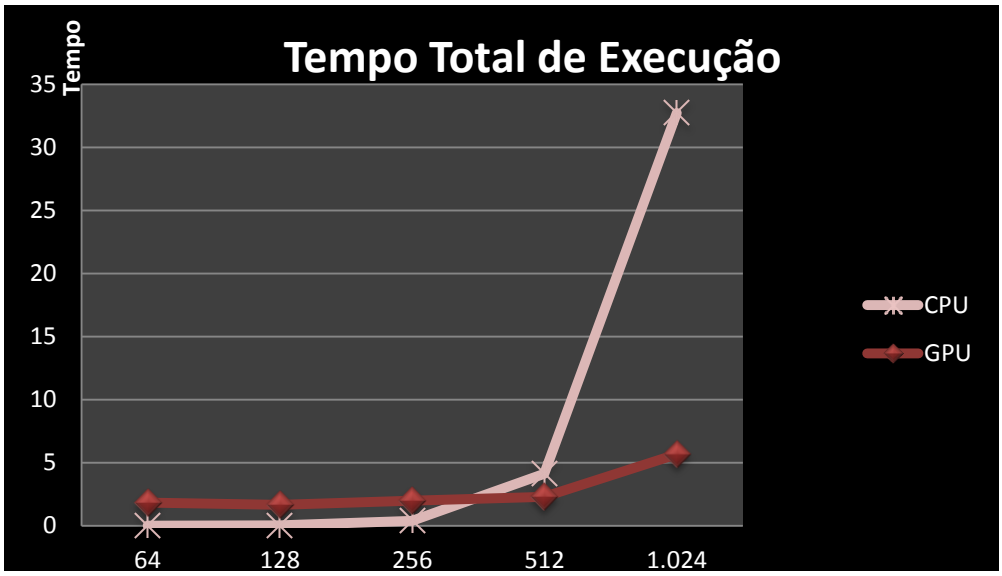


**Gráfico 1 – Tempo Total de Execução (GPUxGPU)**

Para concluir, considerei simples o processo de paralelização por GPU, claro que não podemos comparar com MPI, mas das quatro implementações do Jogo da Vida esta foi a que levei menos tempo, e que os conceitos pensados inicialmente para solucionar o problema não tiveram que ser reestudados (não errei muito).

Na verdade os passos para a paralelização em CUDA apresentados em aula foram muito intuitivos para a solução desse problema, o que de certa forma facilitou muito. Gostei muito do trabalho, acredito que em algum momento isso poderia ser útil para paralelizar rotinas de operações primitivas em matrizes e vetores de forma a melhorar o desempenho em softwares para usuários que disponham de GPU em sua maquina.