# **Enunciado do Projeto (EP5) - Otimização Avançada de Multiplicação de Matrizes com CUDA**

## **Objetivo**

Este projeto tem como objetivo aprofundar os conhecimentos em programação paralela com CUDA, indo além da implementação funcional de um algoritmo. O aluno deverá implementar um kernel para multiplicação de matrizes e, subsequentemente, aplicar técnicas avançadas de otimização para maximizar o desempenho. A etapa final consiste em utilizar ferramentas de profiling profissional (nsys) para analisar, validar e reportar o impacto das otimizações implementadas.

## **Ponto de Partida**

O código base para este projeto está disponível no seguinte link:

* **Arquivo Base:** [01-matrix-multiply-2d.cu no GitHub](https://github.com/HPCSys-Lab/HPC-101/blob/main/problems/01-matrix-multiply-2d.cu)

Este arquivo contém uma função host matrixMulCPU funcional e o esqueleto para o kernel matrixMulGPU que você deverá desenvolver.

## **Tarefa Principal: Implementação e Otimização**

### **Parte 1: Implementação do Kernel Base**

Sua primeira tarefa é implementar o kernel matrixMulGPU de forma que ele produza resultados corretos.

* **Mapeamento 2D:** Utilize uma grade de blocos (grid) e blocos de threads (block) bidimensionais.
* **Um Thread por Elemento:** Projete seu kernel de modo que cada thread seja responsável pelo cálculo de um único elemento da matriz de resultado C.
* **Indexação:** Dentro do kernel, estabeleça índices x e y únicos para cada thread, que corresponderão à coluna e à linha do elemento a ser calculado.

### **Parte 2: Otimização Avançada (Requisito Central)**

Uma vez que seu kernel esteja funcional, você deverá refatorar seu código para incorporar as seguintes otimizações de desempenho:

1. **Otimização da Configuração de Lançamento:**
   * **Objetivo:** Garantir a máxima ocupação (occupancy) dos Streaming Multiprocessors (SMs) da GPU.
   * **Ação:** Em seu código main, consulte programaticamente as propriedades da GPU para obter o número de SMs (props.multiProcessorCount). Utilize este valor para calcular um tamanho de grid que seja um múltiplo do número de SMs, garantindo que haja trabalho suficiente para manter todo o hardware ocupado.
2. **Otimização do Uso de Memória Unificada:**
   * **Objetivo:** Eliminar o gargalo causado pela migração de dados "sob demanda" (page-faulting) entre CPU e GPU.
   * **Ação:** Utilize a função cudaMemPrefetchAsync para guiar explicitamente as transferências de memória. Crie um fluxo lógico onde os dados de entrada são pré-carregados para a GPU antes da execução do kernel, e o resultado é pré-carregado de volta para a CPU antes da verificação.

## **Tarefa Final: Profiling e Relatório de Análise**

A etapa mais importante deste projeto é a análise quantitativa do desempenho.

### **1. Medição de Desempenho**

* Meça e compare os tempos de execução da sua versão otimizada na GPU com a versão na CPU para, no mínimo, três tamanhos de matrizes quadradas (sugestão: 512x512, 1024x1024, 2048x2048).
* Calcule o *speedup* (Tempo CPU / Tempo GPU) para cada caso.

### **2. Profiling com NVIDIA Nsight Systems (nsys)**

* Utilize a ferramenta nsys para gerar um relatório de profiling da sua aplicação otimizada.  
  nsys profile --stats=true ./seu\_executavel <tamanho\_da\_matriz>

### **3. Elaboração do Relatório Técnico**

Prepare um relatório em formato .pdf contendo:

a. Identificação: Seu nome completo e N° UFSCAR.

b. Caracterização do Ambiente: Inclua as propriedades da GPU utilizada (Modelo, Nº de SMs, Capacidade de Cômputo, etc.), obtidas através da sua aplicação.

c. Descrição das Estratégias de Otimização: Detalhe CADA uma das estratégias da "Parte 2", explicando não apenas o que foi feito, mas por que aquela otimização melhora o desempenho.

d. Análise dos Resultados:

- Apresente uma tabela com os tempos de execução e o speedup obtido para os diferentes tamanhos de matriz.

- Análise do Relatório nsys: Esta é a seção mais crítica. Anexe screenshots das seções relevantes do relatório do nsys e explique como elas validam suas otimizações. Por exemplo:

- Na seção Kernel Execution Summary, mostre que o tempo de execução é dominado pelo seu kernel.

- Na seção CUDA Memory Operation Summary, demonstre como o uso de cudaMemPrefetchAsync resultou em poucas operações de [CUDA memcpy Unified...] de grande volume, em vez de um comportamento de page-fault.

e. Conclusão: Resuma os ganhos de desempenho obtidos e a eficácia das técnicas de otimização aplicadas.

## **Formato de Entrega**

* Submeta um único arquivo compactado EP5\_Seu\_Nome\_Completo.zip.
* O arquivo deve conter:
  1. O relatório técnico no formato .pdf.
  2. O código-fonte .cu final e otimizado.
  3. Quaisquer outros arquivos ou scripts utilizados.