# Análise de resultados de comparação do método gauss Jacobi

**Gabriela Bairros Porto**[1], **Cesar Augusto Alves Camillo**[1]

[1]Departamento de Informática
Universidade Federal do Paraná (UFPR) - Curitiba, PR - Brasil

gbp16@inf.ufpr.br, caac16@inf.ufpr.br

***Resumo.*** *Tem-se como objetivo apresentar igualdades e diferenças entre duas implementaçõs do métodos de Gauss Jacobi, uma fazendo a utilização de práticas de otimização discutidas em aula.*

## 1. Funcionamento

Para rodar o trabalho, use o seguinte comando:

*sudo python3 script.py*

- Pode ser necessário realizar update ou instalar as bibliotecas python relevantes (pandas, matplotlib, seaborn, numpy) na máquina onde serão realizados os testes.
- Lembrar de setar o diretório para algum ambiente local
- Pode vir a dar erro por diferença entre versões e pode ter a necessidade de modificação das linhas: 67 e 75 para: **['DP [MFLOP/s]', 'AVX DP [MFLOP/s]']**

## 2. Marcadores

Foram realizados marcadores para:

- A função inteira;
- Alocação de memória
- Leitura de dados em memória
- Para a aplicação do método
- Liberação de memória

## 3. Similaridades

- Ambos realizam uso de variaveis locais, alocadas no inicio das funções, ficam na L1 da cache, ou seja, ficam mais próximo da memória, permitindo menos cache miss.
- A leitura dos dois são iguais, visto que não tem muita alterações para ser feitas

## 4. Diferenças

- Alocação demanda muito tempo, e pode ter a heap comprometida, no código otimizado é feito alocação inicial do ponteiro A[0] separado do resto, e utilizado ele para gerar os seguintes ponteiro de vtores usando o calculo de PAD
- Quando realizando a alocação está sendo calculado para fechar mais ocrretamente com o size passado em parametro
- A maior diferença entre o gauss não otimizado e otimizado é o fato que ele está fazendo somas por blocos, fazendo com que o for de calculo terá que percorrer n / 4 vezes invés de n

- Quando está fazendo acessos para leitura ou escrita de dados, utiliza-se ponteiros realizando acesso direto ao conteudo
- Faz a utilização da função memcpy que é modificada para ser o meio mais otimizado de limpar a memória com um dado único em variavel
- Não tendo a necessidade de tanto gerenciamento de ponteiro pelo meio de alocação, a liberação de memória mais rapida, considerando-se que todos os nós se baseiam no A[0], faz com que não se tenha necessidade de um loop para a liberação da memória.

## 5. Observação

Os gráficos do AVX_DP_MFLOPSps não estão apresentando resultados e por causa disso não foram enviadas no trabalho, mas caso rode o script eles continuaram a ser gerados