# MiniEP5

Diogo José Costa Alves (13709881)

27/04/2023

1. **Mostre, com embasamento estatístico, a variação do tempo de execução entre matrix\_dgemm\_0 e sua implementação de matrix\_dgemm\_1. Houve melhora no tempo de execução? Explique o porquê.**

Partindo do caso com matrizes de dimensão N=2048, rodando 31 vezes cada função e registrando o valor médio, é possível perceber um speed up de mais de 10x entres as implementações.

**dgemm\_0: 73,3861 segundos.**

**dgemm\_1: 7,02896 segundos.**

1. Por que 31 um execuções?

Acredito que esse número vem a partir do teorema limite central (TLC), que afirma que a soma variável a aleatória independente se aproxima de uma distribuição normal à medida que o número de amostras aumenta. Como não conhecemos de antemão a distribuição de origem, seguindo a cartilha do TLC, rodamos mais do que 30 experimentos.

1. Por que o speed up de mais de 10x?

Acredito estar relacionado ao melhor acoplamento entre a forma que definimos a matriz na memória e a sequência de acesso aos dados na memória durante execução. Explicando de outra forma, no código, as matrizes são definidas como arrays contíguos de uma dimensão, onde a linha da matriz equivale a um offset multiplicativo no índice do array e a coluna da matriz equivale a um offset aditivo no índice do array.

Sendo assim, elementos em colunas vizinhas estão localizados sequencialmente no array, já elementos em linhas vizinhas estão distantes N\*8 bytes (double = 8 bytes).

Olhando para as implementações, no código original (dgemm\_0), os elementos da segunda matriz são acessados linha a linha. Como disse, esses acessos estão distantes N\*8 bytes entre si, o que pode dificultar o cache de dados. Durante execução do código original foram contabilizadas 14.991.021.916 **L1-dcache-load-misses**, representando **86,17%** de todos os acessos ao cache L1.

Texto

Descrição gerada automaticamente

Figura 1 - Contadores de performance da execução do código original

Na função aprimorada, (dgemm\_1), alteramos a sequência de acesso a segunda matriz na tentativa de melhorar o acoplamento entre a estrutura de dados da matriz e a sequência de leituras. Nesse casso, passamos a acessar os elementos da segunda matriz, não mais linha à linha mas agora coluna à coluna.

Como tentando explicar anteriormente, esses dados estão guardados sequencialmente no array, o que aumenta a possibilidade de já estarem na cache de dados. Durante execução do código aprimorado, foram contabilizadas 1.079.636.604 **L1-dcache-load-misses**, representando **12,29%** de todos os acessos ao cache L1.

Texto

Descrição gerada automaticamente

Figura 2 - - Contadores de performance da execução do código aprimorado

Apesar de não saber explicitamente, os detalhes da arquitetura de cache e possível perceber que a reorganização nos acessos à segunda matriz foi capaz de gerar um impacto significativo na redução de L1-dcache-misses e em consequência no tempo total de execução.

1. **Execute os mesmos experimentos em outra máquina (por exemplo, a de um colega) e verifique se houve mudança no tempo de execução. Eventuais diferenças podem ocorrer devido a diferenças nas especificações das máquinas (por exemplo, tamanho das linhas de cache)**

|  |  |  |
| --- | --- | --- |
|  | **dgemm\_0:** | **dgemm\_1:** |
| Máquina 1 – Ubuntu Server  Intel(R) Core(TM) i5-3570 CPU @ 3.40GHz  Bogomips: 6784.99  L1 Data cache 256 KB  L2 cache 1MB  L3 cache 6 MB | **73,3861 segundos** | **7,02896 segundos** |
| Máquina 2 – Windows WSL  Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz  Bogomips 6384.00  L1 Data cache 128 KB  L2 cache 1MB  L3 cache 6 MB | **68.0525 segundos** | **5,69452 segundos** |

Apesar da máquina 1 ser de uma geração anterior quando comparada a máquina 2 (3ª versus 6ª geração), estava esperando que ela se saísse melhor, já que possui o dobre de cache L1-data-cache (256KB vs 128KB).

Contrariando minhas expectativas, o código executou mais rápido na Máquina 2, mostrando que existem outros componentes da arquitetura que influenciam na performance além do tamanho do cache L1-data.

1. **Comentário sobre os desafios encontrados.**

Gostaria apenas de deixar um pequeno comentário sobre a experiência de desenvolvimento em C.

É muito bom perceber que uma implementação em C pode ter uma performance 10-100x melhor que a do python. Mas, minha falta de fluência na linguagem e ter compilar executar o código manualmente acabou deixou o processo de experimentação um pouco maçante.

Vou dar uma pesquisada em algo para compilar e rodar o código automaticamente no editor, mas se tiverem alguma dica para melhorar a experiência já agradeço muito.