UNIVERSIDADE DO VALE DO ITAJAÍ
CIÊNCIA DA COMPUTAÇÃO

ORGANIZAÇÃO DE COMPUTADORES

HIERARQUIA DE MEMÓRIA

PROFESSOR THIAGO FELSKI PEREIRA
ALUNOS: DANIEL SANSÃO ARALDI, RAFAEL MOTA ALVES E JONATHAS MEINE

ITAJAÍ
JUNHO, 2024

Introdução

Este relatório visa demonstrar o desempenho das diferentes possibilidades de organizações de cache com 1024 Bytes. Para os testes, foi utilizado um programa em Assembly da arquitetura RISC-V de 32 bits, o qual executa a adição de duas matrizes 10x10 e armazena o resultado em uma terceira matriz. Neste programa, é preciso escolher o método linha-coluna ou coluna-linha para percorrer as matrizes, que foi um dos principais critérios da análise feita.

Programa Assembly

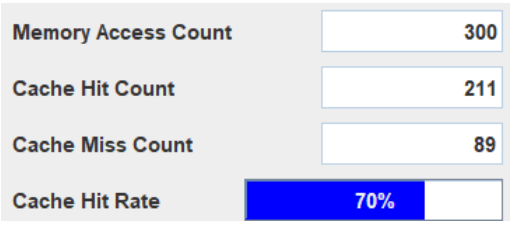
O programa Assembly utilizado pode ser encontrado no repositório público do GitHub através do link https://github.com/jonhymeine/sum-of-matrices.

Resultados

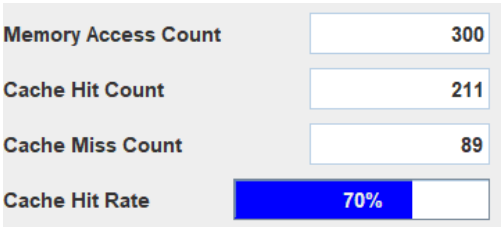
Para os resultados, foram apenas consideradas as execuções das somas de matrizes, não incluindo a interação inicial do usuário e a impressão da matriz resultado. Para cada organização, foram simulados ambas as opções de iteração: linha-coluna e coluna-linha, trazendo a contagem de acertos e faltas na cache com 300 acessos à memória.

1 bloco de 256 *words*

LINHA-COLUNA

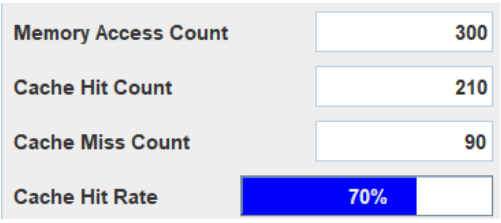


COLUNA-LINHA

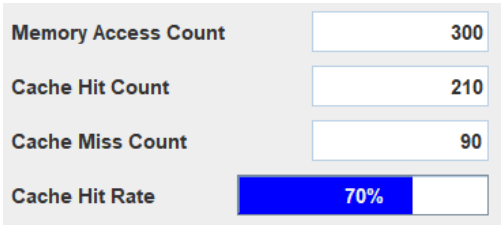


2 blocos de 128 *words*

LINHA-COLUNA



COLUNA-LINHA



4 blocos de 64 *words*

LINHA-COLUNA

Memory Access Count	300
Cache Hit Count	279
Cache Miss Count	21
Cache Hit Rate	93%

COLUNA-LINHA

Memory Access Count	300
Cache Hit Count	261
Cache Miss Count	39
Cache Hit Rate	87%

8 blocos de 32 *words*

LINHA-COLUNA

Memory Access Count	300
Cache Hit Count	290
Cache Miss Count	10
Cache Hit Rate	97%

COLUNA-LINHA

Memory Access Count	300
Cache Hit Count	254
Cache Miss Count	46
Cache Hit Rate	85%

16 blocos de 16 *words*

LINHA-COLUNA

Memory Access Count	300
Cache Hit Count	281
Cache Miss Count	19
Cache Hit Rate	94%

COLUNA-LINHA

Memory Access Count	300
Cache Hit Count	227
Cache Miss Count	73
Cache Hit Rate	76%

32 blocos de 8 *words*

LINHA-COLUNA

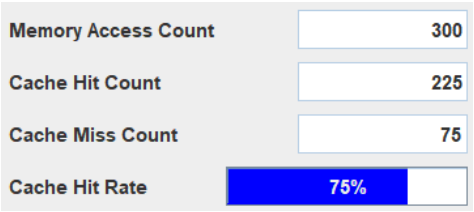
Memory Access Count	300
Cache Hit Count	262
Cache Miss Count	38
Cache Hit Rate	87%

COLUNA-LINHA

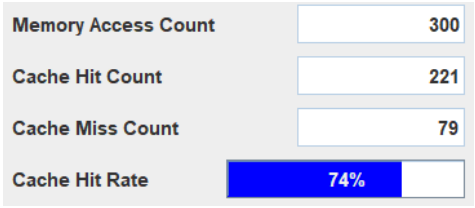
Memory Access Count	300
Cache Hit Count	208
Cache Miss Count	92
Cache Hit Rate	69%

64 blocos de 4 *words*

LINHA-COLUNA

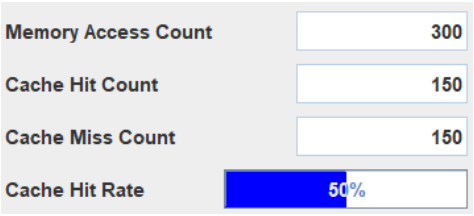


COLUNA-LINHA

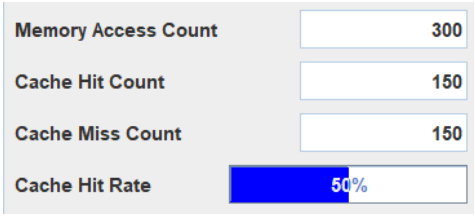


128 blocos de 2 *words*

LINHA-COLUNA

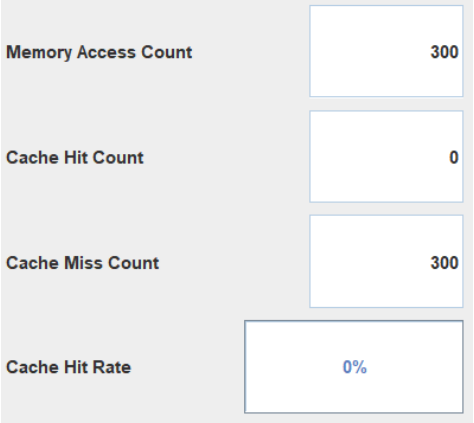


COLUNA-LINHA

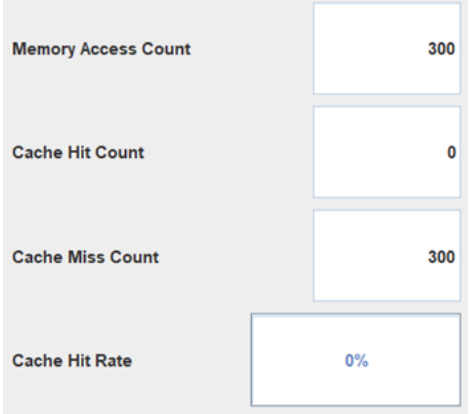


256 blocos de 1 *word*

LINHA-COLUNA



COLUNA-LINHA



Análise

Com os resultados obtidos, foi observado que, a partir de 4 blocos, é possível encontrar uma grande melhora na taxa de acertos, isso porque, no programa, são manipuladas 3 matrizes de 100 valores simultaneamente, assim, sendo necessário pelo menos 3 blocos para desempenhar bem, para que não haja a necessidade constante de substituir blocos que ainda são necessários, destacando as organizações de 4 e 8 blocos.

Logo, foram identificados que, para linha-coluna, a melhor organização foi a de 8 blocos de 32 *words* com 97% de acertos e, para coluna-linha, a melhor foi a de 4 blocos

de 64 *words* possuindo uma taxa de acertos de 87%. Já, para ambos os métodos, a pior simulação foi a de 256 blocos de apenas uma palavra, resultando em uma porcentagem de acertos de 0%.

Em uma organização de cache de 8 blocos de 32 palavras, promove-se uma localidade temporal pequena por conta da quantidade de blocos, entretanto, uma localidade espacial mais significativa devido ao tamanho dos blocos. Assim, para o método linha-coluna, foi obtida uma grande taxa de acertos na cache devido à exploração da localidade espacial dos blocos da cache.

Já, para a organização de cache de 4 blocos de 64 palavras, encontra-se uma localidade espacial alta, e uma localidade temporal mais baixa. Logo, para o método coluna-linha, que não acessa valores sequenciais, são necessários blocos grandes para poder acertar na cache e, assim, usufruir da localidade espacial.

Agora, para a organização de cache de 256 blocos de uma palavra, promove-se uma alta localidade temporal por ter uma grande quantidade de blocos, mas, por ter apenas uma palavra por bloco, apresenta uma baixíssima localidade espacial. No programa, a localidade temporal não é usufruída, porque não há repetição de valores de memória já utilizados, e, como cada bloco possui apenas uma palavra, não é possível explorar a localidade espacial.