Avaliação de desempenho de um único núcleo

Faculdade de Engenharia da Universidade do Porto
Carolina Ferreira, up201905810
Maria Miguel Ribeiro, up201906945
Pedro Santos, up201907254

Introdução

Este projeto tem como finalidade o estudo do efeito no desempenho do processador da hierarquia da memória ao aceder a grandes quantidades de dados, utilizando a multiplicação de matrizes e a API de desempenho (PAPI) para avaliar o desempenho da execução do programa. Ao longo do relatório são explicados os algoritmos implementados e serão discutidos os resultados obtidos.

Exercício 1

Para este exercício, o pretendido é a implementação do código fornecido em C++, para Java. Este código multiplica duas matrizes. Para um bom funcionamento, foi substituida a função clock() pertencente à estrutura do SYSTEMTIME pela função currentTimeMillis() de java. As funcionalidades do papi não são aplicáveis ao código em java.

Exercício 2

Neste exercício, com o objetivo de implementar uma versão que multiplica um elemento da primeira matriz, a, pela linha correspondente da segunda matriz, b, criamos 3 ciclos for. Desta forma, o primeiro elemento de cada coluna da matriz a é multiplicado por todos os elementos da primeira linha de b. Voltando ao inicio da linha da matriz c, somamos ao valor anterior de cada posição, o valor obtido da multiplicação do primeiro elemento da segunda coluna da matriz a, com cada elemento da segunda linha da matriz b, e assim sucessivamente de forma a obtermos a primeira linha da matriz c (matriz resultante da multiplicação), as restantes linhas são obtidas da mesma forma, mas utilizando o elemento n de cada coluna da matriz a para obter a linha n da matriz c.

Exercício 3

Este exercício visa a implementação de código que, divida as matrizes a e b numa quantidade de blocos indicada inicialmente pelo utilizador e multiplique cada submatriz da matriz a pela submatriz correspondente da matriz b utilizando um dos métodos de multiplicação definidos nos exercícios anterios, obtendo a submatriz correspondente da matriz c.

Métricas de avaliação da Performance.

Para avaliar o desempenho do processador nos diferentes algoritmos utilizamos as metricas L1_DCM (Level 1 data cache misses), L2_DCM (Level 2 data cache misses), PRF_DM (Prefetch data instruction caused a miss), TLB_DM (Data translation lookaside buffer misses), TOT_INS (Total instructions executed) e TOT_CYC (Total cycles). Utilizamos os contadores L1_DCM e L2_DCM que registam os caches misses dos dados, pois têm bastante impacto no desempenho do processador. Este começa por aceder à cache L1 onde ocorre um hit caso os dados solicitados estejam disponíveis na cache, caso contrário ocorre um cache miss e vai para a cache 2. Uma falha em alguma das caches faz com que ocorra a procura de dados nos sistemas de memória, aumentando o custo.

A métrica PRF_DM foi usada devido à técnica utilizada para acelerar a performance do processador dado que escreve numa memória mais rápida dados do armazenamento original, mesmo antes de serem necessários.

Escolhemos usar o parâmetro TLB_DM dado que simplifica a tradução de endereços lineares em endereços físicos, e desta forma aumenta o desempenho do processador porque evita a consulta à tabela de páginas localizada em memória.

As métricas TOT_INS e TOT_CYC foram utilizadas pois permitem observar o desempenho do processador através dos MIPS e dos FLOPS.

Resultados e análises.

Exercício 1.

Os resultados obtidos mostram que numa matriz de dimensão 600x600, a quantidade de cache misses é relativamente baixa. À medida que o tamanho das matrizes vai aumentando notamos um aumento exponencial do número de cache misses tanto na cache L1 como na L2. Comparando então as duas caches, verificamos que começam por ter mais ou menos o mesmo desempenho mas com um tamanho de matriz NxN a cima de 2000, a cache L1 apresenta um melhor desempenho. Podemos também verificar que a razão entre L2 e L1 vai aumentando à medida que o tamanho da matriz aumenta.

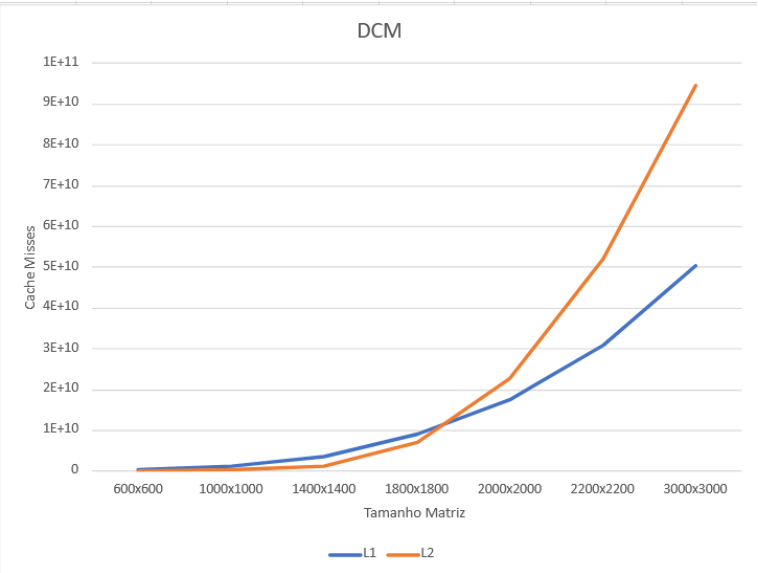
Para métricas PRF_DM e TLB_DM notamos que em ambas à medida que a dimensão da

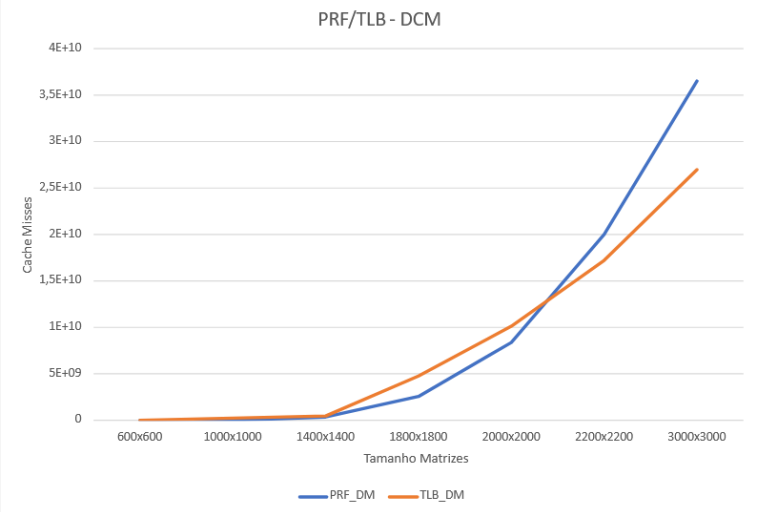
matriz aumenta o número de data misses aumenta também. Ao longo do tempo de execução verificamos que os valores das duas métricas foram semelhantes.

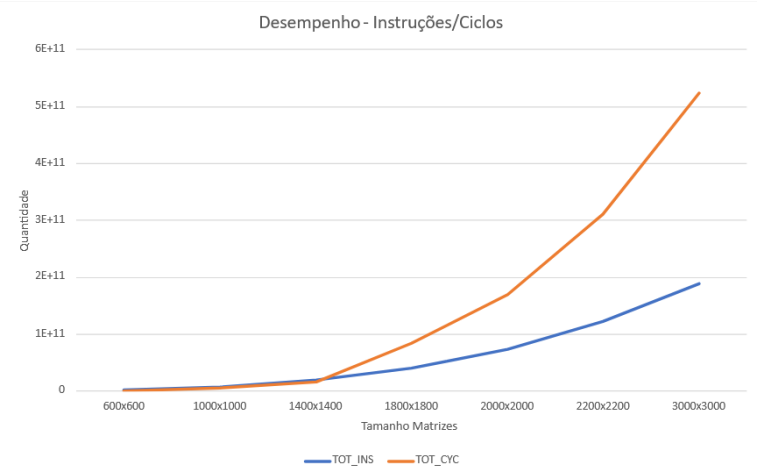
No parâmetro TOT_INS constatamos também que o total de instruções executadas aumentava com o crescimento da dimensão da matriz.

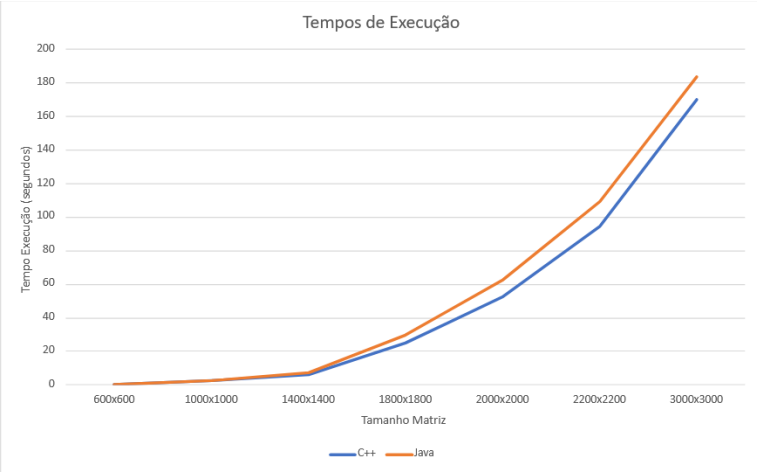
Para a métrica TOT_CYC vimos que tal como nos restantes parâmetros, para uma dimensão da matriz maior o número total de ciclos aumentava.

Quanto ao tempo de execução em C/C++ e em Java, observamos que o código em Java demorou mais tempo a executar.









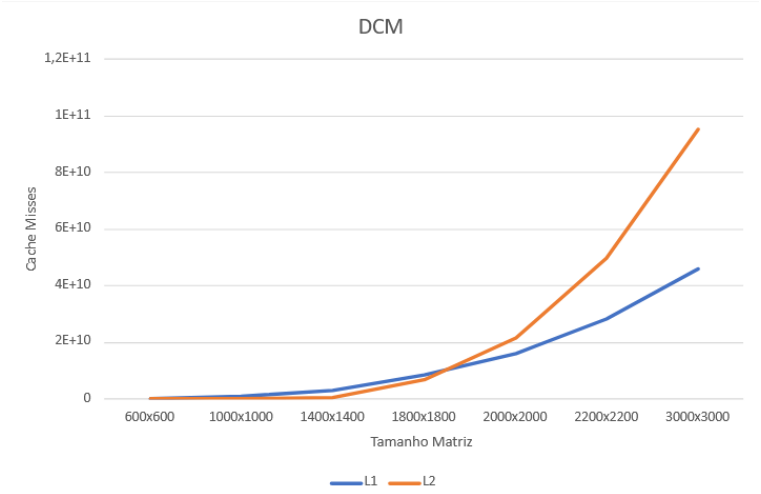
Exercício 2.

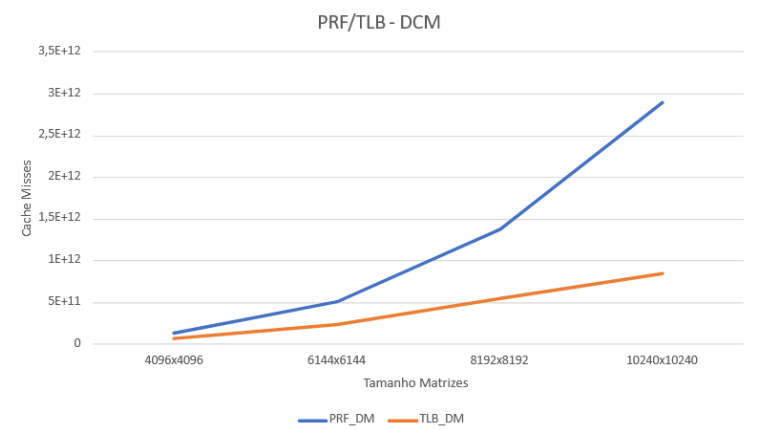
Os resultados obtidos com o código implementado no exercício 2 são semelhantes aos que foram obtidos no exercício 1. A cache L1 demonstra um melhor desempenho do que L2, mas ambas num crescimento acentuado de cache misses tal como no exercício 1.

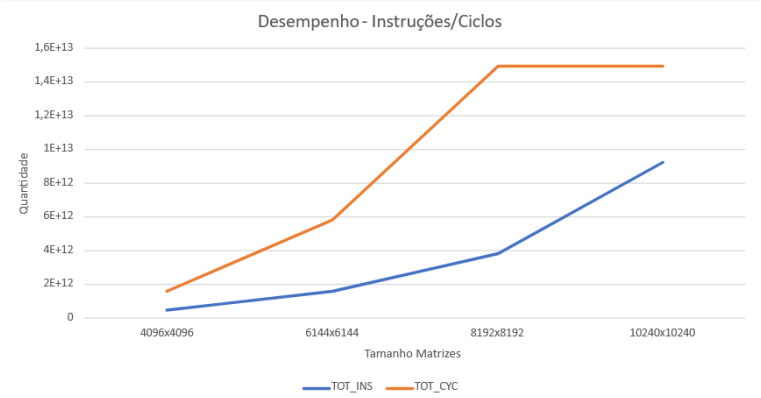
Á semelhança do exercício anterior nas métricas PRF_DM e TLB_DM, notamos que os valores de data misses aumentavam com o aumento da dimensão da matriz.

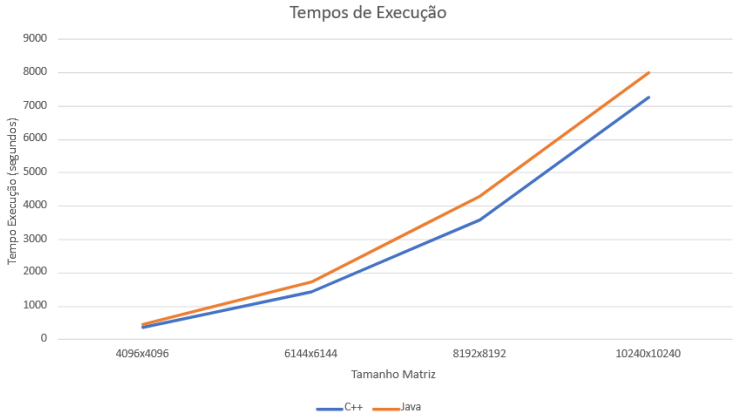
Para os parâmetros TOT_INS E TOT_CYC verificamos também o aumento dos valores consoante o aumento da dimensão da matriz.

Quanto ao tempo de execução em C/C++ e em Java, concluímos o mesmo que no exercício anterior, o código em Java demorou mais tempo a executar.


DCM
1,2E+11
1E+11
8E+10
6E+10
4E+10
2E+10
0
Cache Misses
600x600
1000x1000
1400x1400
1800x1800
2000x2000
2200x2200
3000x3000
Tamanho Matriz
L1
L2

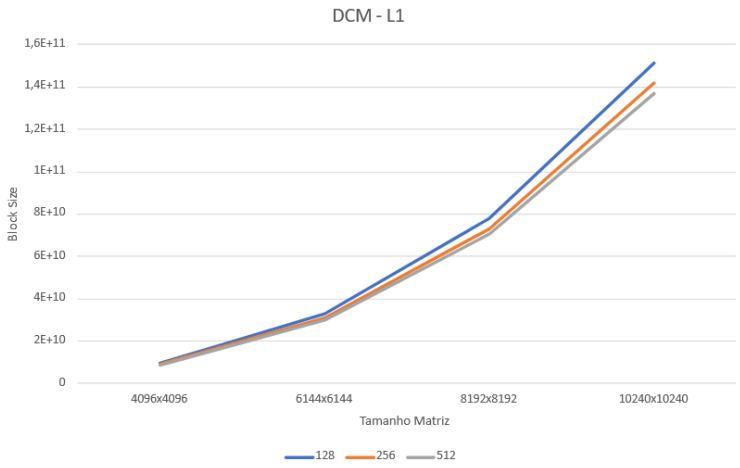

PRF/TLB - DCM
3,5E+12
3E+12
2,5E+12
2E+12
1,5E+12
1E+12
5E+11
0
Cache Misses
4096x4096
6144x6144
8192x8192
10240x10240
Tamanho Matrizes
PRF_DM
TLB_DM

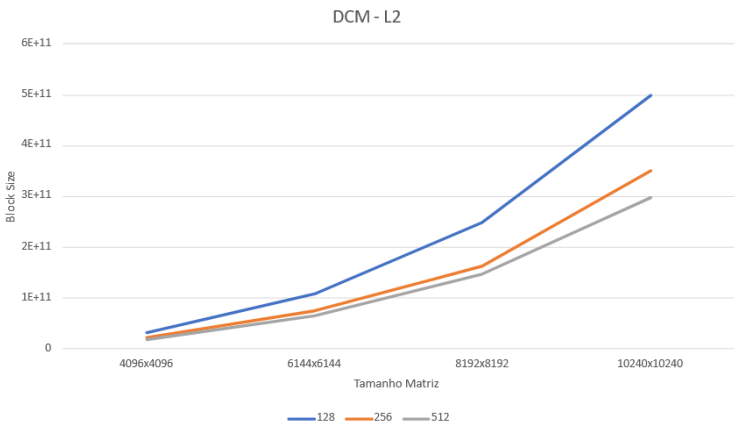

Desempenho - Instruções/Ciclos
1,6E+13
1,4E+13
1,2E+13
1E+13
8E+12
6E+12
4E+12
2E+12
0
Quantidade
4096x4096
6144x6144
8192x8192
10240x10240
Tamanho Matrizes
TOT_INS
TOT_CYC

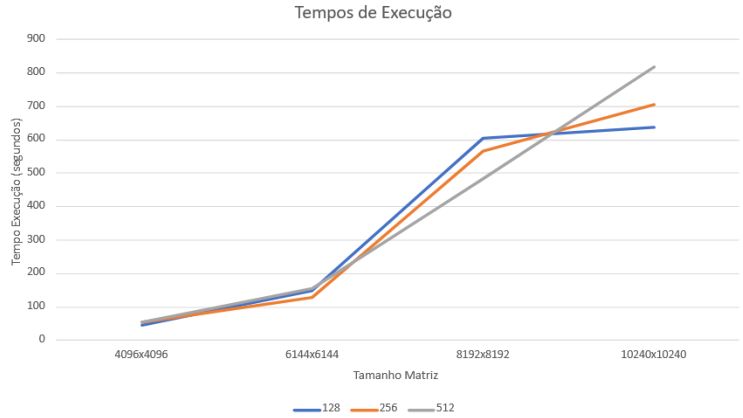


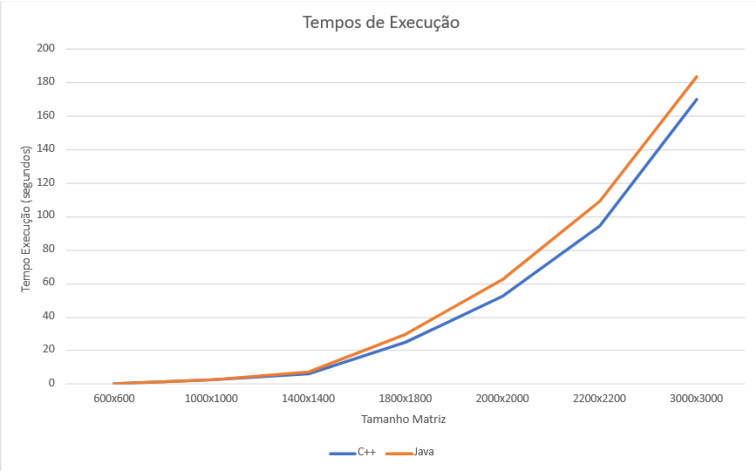
Exercício 3.

Observando os gráficos resultantes da aplicação do algoritmo OnMultBlock para blocos de diferentes tamanhos observamos que de maneira geral para blocos de 512 obtivemos uma menor quantidade de cache misses em comparação com os blocos de 256 e 128 elementos. Relativamente ao tempo de execução verificamos que para blocos de 128 elementos o tempo de execução tinha um crescimento muito menos acentuado assim que as matrizes atingiram um tamanho de 8192x8192.









Após a observação dos resultados obtidos para cada um dos algoritmos constatamos que o números de caches misses era superior no algoritmo onMultBlock comparadamente aos outros algoritmos. Para as caches L1 e L2 o algoritmo do exercício 2 apresenta menor número de caches misses do que o exercício 1. Para as métricas PRF_DM, TLB_DM, TOT_INS e TOT_CYC os valores foram superiores no algoritmo do exercício 2. Concluimos então que o algoritmo onMultBlock foi o que apresentou melhor desempenho, dado que a divisão em blocos facilita o acesso aos dados.

Conclusão

O desenvolvimento deste projeto permitiu-nos perceber melhor o desempenho do processador ao aceder a grandes quantidades de dados bem como das métricas utilizadas para avaliação do mesmo e desta forma adquirir conhecimentos sobre Performance API.

Confrontámo-nos com algumas dúvidas na implementação dessas métricas e com alguma dificuldade devido aos elevados tempos de execução para uma grande quantidade de dados, mas que foram rápidamente superadas.

Embora as adversidades, concluímos que o grupo conseguiu elaborar o trabalho prático com sucesso.

Referências

- Pesquisa das métricas:
  - https://www.cs.uoregon.edu/research/tau/docs/newguide/bk03ch01s04.html