Faculdade de Engenharia da Universidade do Porto

Computação Paralela e Distribuída

1º Relatório

Grupo 02

Estudantes:

Nuno França up201807530@up.pt

David Fang up202004179@up.pt

Miguel Tavares up202002811@up.pt

Índice

Problema e algoritmos

Descrição do problema

Este projeto foi realizado no âmbito da Unidade Curricular de Computação Paralela e Distribuída, onde foi estudado o efeito de desempenho do processador na hierarquia de memória quando acedidas enormes quantidades de informação. Neste estudo, foi usado o produto de duas matrizes. Recorrendo ao auxílio da *Perfomance API (PAPI)*, foram colecionados indicadores da execução do programa.

Para este estudo, foram implementados três algoritmos de multiplicação de matrizes e registaram-se dados para a análise. Foram realizadas duas implementações para os dois primeiros algoritmos, uma em C++ e outra em Java, de forma a comparar diferentes aspetos em termos de desempenho.

Algoritmos

Nesta secção serão apresentados os algoritmos abordados no projeto: multiplicação *naive*, por linhas e por blocos de matrizes.

Multiplicação *Naive* de matrizes

```
for i = 1 to N do
      for j = 1 to N do
            for k = 1 to N do
                  C[i, j] += A[i, k] * B[k, j]
            endfor
      endfor
endfor
```

Representação do algoritmo de Multiplicação *naive* em pseudocódigo

Este algoritmo recorre à definição de multiplicação de matrizes, originando ciclos ***nested*** com complexidade temporal de **$O(N^3)$** que percorre de linha a linha da matriz **A** e de coluna a coluna da matriz **B**, guardando o resultado na matriz **C**.

É esperado que o número de acessos à memória cache seja deduzida pela expressão:

$$m(N) = N^3 + N^2 + 2N^2$$

Multiplicação por linhas

```
for i = 1 to N do
      for k = 1 to N do
            for j = 1 to N do
                  C[i, j] += A[i, k] * B[k, j]
            endfor
      endfor
endfor
```

Representação do algoritmo de Multiplicação por linhas em pseudocódigo

Em comparação ao primeiro algoritmo, a forma como é implementada é bastante semelhante à multiplicação ***naive***, permanecendo a mesma complexidade do tempo e do número de acessos à memória cache.

O algoritmo foi alterado de maneira a que a iteração na matriz seja feita pelas suas linhas. Iterando as linhas das matrizes é realizada a soma dos seus valores na matriz **C**.

Multiplicação por blocos

```
for i from 1 to N by BlockSize do
    for j from 1 to N by BlockSize do
        for k from 1 to N by BlockSize do
            for ii from i to min(i + BlockSize, N) do
                 for kk from k to min(k + BlockSize, N) do
                    for jj from j to min(j + BlockSize, N) do
                        c[ii, jj] += A[ii, kk] * B[kk, jj]
                    endfor
                endfor
            endfor
        endfor
    endfor
endfor
```

Representação do algoritmo de Multiplicação por blocos em pseudocódigo

Diferentemente dos dois primeiros algoritmos, a multiplicação por blocos divide as matrizes **A** e **B** em blocos e recorre à multiplicação por linhas para a obtenção da matriz resultante, **C**.

É esperado que o número de acesso à memória seja deduzida através da seguinte expressão:

$$
\begin{aligned}
m(N) = {} & a)\ N^3 * (n^2 / N^2) \\
& b)\ N^3 * (n^2 / N^2) \\
& c)\ 2 * N^2 * (n^2 / N^2) \\
= {} & (2 * N + 2) * n^2
\end{aligned}
$$

Valores de Desempenho

Em **C++**, a partir da dimensão da matriz, obteve-se o **tempo** de processamento correspondente a cada um dos diferentes algoritmos, os respetivos valores de ***Data Cache Misses*** nos níveis **1** e **2** do processador e o número de **instruções** executadas no processador, enquanto em **Java**, apenas foi registado o **tempo** de processamento.

Relativamente às duas linguagens referidas, espera-se que o tempo de execução seja inferior quando o programa é executado em **C++**, devido ao facto de ser uma linguagem de programação compilada, diferentemente de Java que compila o código fonte em ***bytecode*** para que seja interpretada pelo ***Java VM***.[1]

Prevê-se que pela ordem respectiva de eficiência de algoritmos, que a multiplicação por blocos seja a mais eficiente, devido a uma maior utilização da cache na resolução de submatrizes de pequenas dimensões, em seguida, na multiplicação por linhas existe maior probabilidade de ocorrer ***Data Cache Miss*** devido às linhas das matrizes de maiores dimensões, por último, multiplicação ***Naive***, devido a um número elevado de acessos à memória em busca dos valores que se encontram nos matrizes **A** e **B**, isto resulta numa fraca utilização da cache porque o ***CPU*** tem de ir buscar constantemente novas linhas de cache para aceder ao próximo conjunto de elementos, o que origina uma taxa de ***Data Cache Misses*** mais elevada aos algoritmos anteriores.

[1] "C++ vs Java - Javatpoint." https://www.javatpoint.com/cpp-vs-java.

Resultados e análise

Os resultados obtidos nos exercícios seguintes foram registados a partir de um computador portátil com a seguinte especificação de processador: I7-8750H.

Exercício 1

Size	Time(s) C++	Time(s) java	Level 1 DCM	Level 2 DCM	No. INS
600	0.439	0.526	2.45E+08	3.99E+07	8,61E+08
1000	1.330	5.043	1.22E+09	3.00E+08	5,53E+09
1400	3.941	16.954	3.42E+09	1.46E+09	1,64E+10
1800	20.420	40.137	9.09E+09	7.84E+09	8,14E+10
2200	43.018	87.299	1.76E+10	2.20E+10	1,75E+11
2600	77.935	143.43	3.09E+10	5.12E+10	3,14E+11
3000	129.709	244.532	5.03E+10	9.71E+10	5,26E+11

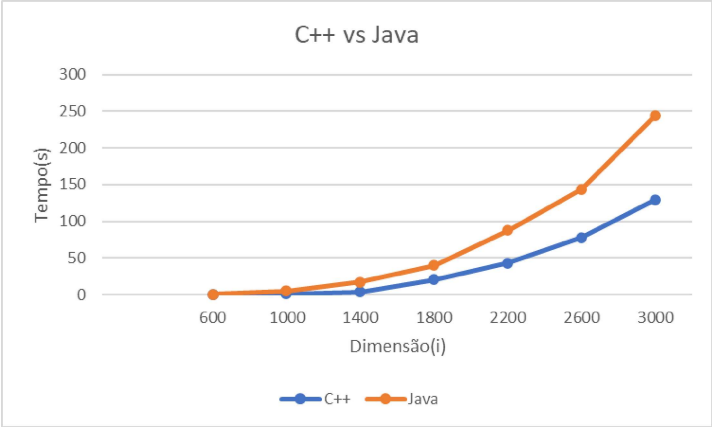

Figura 1 - Multiplicação *naive* de matrizes (C++ vs java)

Após a análise de dados recolhidos, é destacado um ponto de divergência que surge próximo do valor de dimensão de 1000 e também foi apontado que o tempo varia com o aumento da dimensão.

Exercício 2 a)

Size	Time(s) C++	Time(s) java	Level 1 DCM	Level 2 DCM	No. INS
600	0.337	0.338	2.71E+07	5.61E+07	4,58E+08
1000	0.594	1.559	1.26E+08	2.57E+08	2,40E+09
1400	1.794	1.846	3.46E+08	7.00E+08	7,27E+09
1800	3.857	3.874	7.45E+08	1.49E+09	1,56E+10
2200	7.56	7.356	2.08E+09	2.70E+09	2,86E+10
2600	11.68	11.889	4.41E+09	4.45E+09	4,74E+10
3000	17.937	18.583	6.78E+09	6.82E+09	7,43E+10

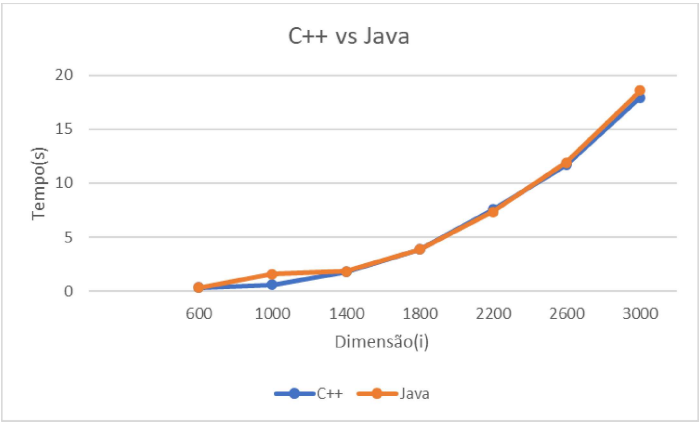

Figura 2a - Multiplicação por linhas (C++ vs java)

Em comparação ao primeiro exercício, foi anotado uma melhor performance na execução do algoritmo, devido ao menor **tempo de processamento** e número de ***Data Cache Misses*** registados, o que induz a um maior aproveitamento da cache durante a execução.

Comparando as linguagens de programação utilizadas, foi apontada uma semelhança entre os valores obtidos, o que não corresponde ao previsto na secção anterior, “Valores de Desempenho”.

Exercício 2 b)

Size	Time(s) c++	Time(s) java	Level 1 DCM	Level 2 DCM	No. INS
4096	45.947	105.61	1.77E+10	1.77E+10	1,84E+11
6144	154.789	379.905	5.96E+10	5.96E+10	6,20E+11
8192	367.396	908.857	1.41E+11	1.41E+11	1,48E+12
10240	717.455	1771.236	2.76E+11	2.80E+11	2,90E+12

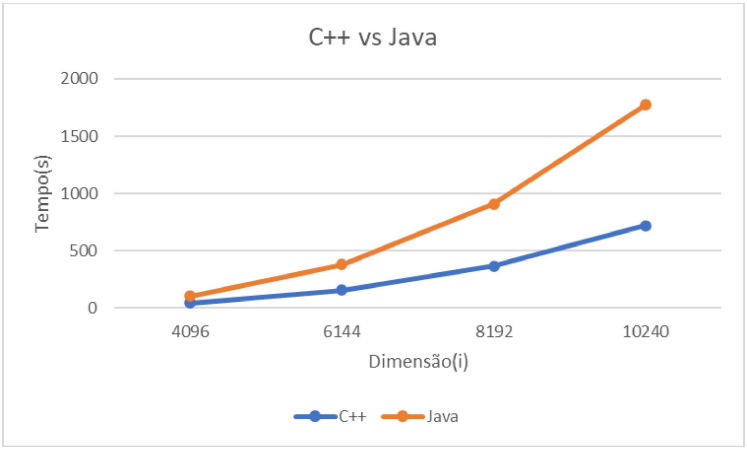

Figura 2b - Multiplicação por linhas

Embora os valores de tempo denotam uma grande diferença entre **C++** e **Java**, foi possível identificar um crescimento idêntico a partir da dimensão de 6144, em que, com o aumentar da dimensão, a razão entre o tempo em **Java** e o tempo em **C++** aproxima-se de uma constante.

Exercício 3

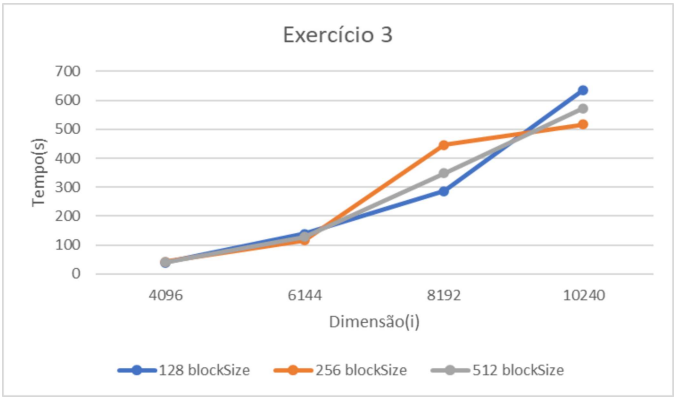

Figura 3 - Multiplicação por blocos

	Block Size 128			Block Size 256			Block Size 512		
Size	Time(s) C++	Level 1 DCM	Level 2 DCM	Time(s) C++	Level 1 DCM	Level 2 DCM	Time(s) C++	Level 1 DCM	Level 2 DCM
4096	40.036	9.73E+09	3.17E+10	42.319	9.10E+09	2.23E+10	41.288	8.76E+09	1.90E+10
6144	139.348	3.28E+10	1.07E+11	115.796	3.07E+10	7.53E+10	128.161	2.96E+10	6.49E+10
8192	286.599	7.78E+10	2.51E+11	446.473	7.30E+10	1.67E+11	348.344	7.02E+10	1.49E+11
10240	636.175	1.52E+11	4.94E+11	516.992	1.42E+11	3.49E+11	572.690	1.37E+11	3.00E+11

Neste exercício, concluiu-se que os dados temporais evoluíram de forma semelhante como nos exercícios anteriores com o crescimento da dimensão, embora não foi apontado nenhuma predominância de resultado sob qualquer dimensão da matriz ou tamanho dos blocos.

Giga Floating Point Operations per Second (GFLOPS)

Após a análise dos resultados obtidos, foi possível verificar que o **número de operações por segundo** nos diversos exercícios apresentados anteriormente manteve-se num valor de 4,0 **GFLOPS** com um ligeiro desvio padrão.

Conclusões

Em resumo, o desenvolvimento deste trabalho permitiu uma melhor perceção de conceitos lecionados nesta unidade curricular e os resultados de desempenho face aos diferentes algoritmos e à complexidade destes, tanto temporal como espacial.