

# Performance de C, LISP e JAVA no cálculo do produto de duas matrizes

## Objectivo

Com este trabalho pretendemos testar a performance no cálculo do produto de duas matrizes das linguagens C, LISP e JAVA. Para tal definimos um algoritmo que calcula o produto de duas matrizes quadradas e procedemos à sua implementação nas três linguagens.

## Descrição do algoritmo utilizado

O algoritmo utilizado é o mesmo para as 3 linguagens e consiste em 4 passos:

1. criação de três matrizes quadradas  $n \times n$ ;
2. inicialização da primeira matriz, colocando em todos os campos o valor “12”;
3. inicialização da segunda matriz, colocando em todos os campos o valor “12”;
4. inicialização da terceira matriz, colocando em todos os campos o valor “0”;
5. cálculo do produto da primeira matriz pela segunda, sendo o resultado guardado na terceira matriz.

Observações: Os campos da primeira e da segunda matriz foram inicializados com um valor constante, visto que o que queremos testar é a velocidade do cálculo do produto e não a criação de números aleatórios. Esse valor foi escolhido de forma aleatória.

## Implementação e realização dos testes

Visto que o objectivo é testar a performance do algoritmo, nenhum dos programas está optimizado ao máximo, sendo todos equivalentes. Os testes foram realizados na máquina “mega”, visto que todos os interessados têm acesso à mesma. É de salientar que o mega é uma máquina multiutilizador, o que faz com que os testes não sejam os mais correctos, visto que não somos os únicos a utilizar tempo de processador. Sendo assim o que nos interessa são os tempos relativos e não os tempos absolutos. Os tempos referem-se ao tempo de utilizador gasto.

### teste ao ficheiro matrix.c:

```
$ gcc -O3 matrix.c  
$ time a.out
```

### teste ao ficheiro matrix.java:

```
$ javac -O matrix.java  
$ time java matrix
```

### teste ao ficheiro matrix.lisp:

```
$ lisp  
* (compile-file “matrix.lisp”)  
* (load “matrix.sparcf”)  
* (time (foo matrix_1 matrix_2 matrix_3))  
* (quit)
```

## Resultados obtidos

Realizámos dois lotes de testes, um à noite e outro de manhã, e repetimos cada teste três vezes. Utilizámos três valores diferentes para n de modo a podermos ter uma visão mais alargada da performance. O formato de tempo está definido como <minutos:segundos>.

Resultados obtidos às 3h:

valor de n	matrix.c	matrix.lisp	matrix.java
128	0:00,17	0:00,20	0:00,87
	0:00,18	0:00,21	0:00,87
	0:00,18	0:00,21	0:00,87
512	0:15,51	0:15,05	0:29,28
	0:15,45	0:15,12	0:29,29
	0:15,43	0:15,09	0:29,32
1024	2:42,65	2:18,59	13:37,31
	2:44,08	2:18,88	13:36,90
	2:42,86	2:19,72	13:35,86

Resultados obtidos às 11h:

valor de n	matrix.c	matrix.lisp	matrix.java
128	0:00,19	0:00,21	0:00,91
	0:00,18	0:00,20	0:00,92
	0:00,18	0:00,22	0:00,90
512	0:15,38	0:15,18	0:32,53
	0:15,42	0:15,16	0:30,72
	0:15,56	0:15,25	0:30,90
1024	2:48,01	2:44,61	14:54,53
	2:50,91	2:44,76	14:56,21
	2:50,65	2:44,73	14:55,14

## Conclusão

Os resultados mostram-nos que tanto o C como o LISP têm tempos praticamente equivalentes, enquanto que, neste caso, JAVA é significativamente mais lento. Muito provavelmente isto deve-se ao facto de o programa em JAVA estar a correr sobre uma máquina virtual. Podemos concluir que tanto C como LISP têm a mesma potencialidade para resolver este género de problemas, que consiste basicamente em ciclos, somas, multiplicações e acessos à memória.

Henrique Rodrigues, nº 48266  
Tiago Maduro-Dias, nº 48399

## Anexo 1 - Código do programa matrix.c

```
#define DIM 1024

unsigned int matrix_1[DIM][DIM];
unsigned int matrix_2[DIM][DIM];
unsigned int matrix_3[DIM][DIM];

main () {
    int i, j, k;

    /* inicializacao da primeira matriz */
    for (i=0; i<DIM; i++) {
        for (j=0; j<DIM; j++) {
            matrix_1[i][j] = 12;
        }
    }

    /* inicializacao da segunda matriz */
    for (i=0; i<DIM; i++) {
        for (j=0; j<DIM; j++) {
            matrix_2[i][j] = 12;
        }
    }

    /* inicializacao da terceira matriz */
    for (i=0; i<DIM; i++) {
        for (j=0; j<DIM; j++) {
            matrix_3[i][j] = 0;
        }
    }

    /* calculo do produto de duas matrizes quadradas */
    for (i=0; i<DIM; i++) {
        for (j=0; j<DIM; j++) {
            for (k=0; k<DIM; k++) {
                matrix_3[i][j] = matrix_3[i][j] + (matrix_1[k][j] * matrix_2[i][k]);
            }
        }
    }
}
```



## Anexo 3 - Código do programa matrix.java

```
import java.util.*;

public class matrix {
    public static void main (String[] args) {

        int i, j, k;
        final int dim = 1024;

        int[][] matrix_1 = new int[dim][dim];
        int[][] matrix_2 = new int[dim][dim];
        int[][] matrix_3 = new int[dim][dim];

        /* inicializacao da primeira matriz */
        for (i=0; i<dim; i++) {
            for (j=0; j<dim; j++) {
                matrix_1[i][j] = 12;
            }
        }

        /* inicializacao da segunda matriz */
        for (i=0; i<dim; i++) {
            for (j=0; j<dim; j++) {
                matrix_2[i][j] = 12;
            }
        }

        /* inicializacao da terceira matriz */
        for (i=0; i<dim; i++) {
            for (j=0; j<dim; j++) {
                matrix_3[i][j] = 0;
            }
        }

        /* calculo do produto de duas matrizes quadradas */
        for (i=0; i<dim; i++) {
            for (j=0; j<dim; j++) {
                for (k=0; k<dim; k++) {
                    matrix_3[i][j] = matrix_3[i][j] + (matrix_1[k][j] * matrix_2[i][k]);
                }
            }
        }
    }
}
```