***Faculdade de Engenharia da Universidade do Porto***

***Computação Pararela - 4º ano MIEIC – 2012/2013***

Multiplicação de matrizes

*Projeto 1*

02/04/2013

Carlos Miguel Correia da Costa

[**ei09097@fe.up.pt**](mailto:ei09097@fe.up.pt) **/ Carlos.costa@fe.up.pt**

200903044

Descrição do projeto

# Contextualização

**O presente projeto tem como objetivo comparar a implementação de uma mesma funcionalidade usando estratégias de programação diferentes, de forma a salientar que a forma de aumentar a eficiência e rapidez de um sistema começa sempre pelo conhecimento da plataforma onde este irá correr, e só depois é que se deverá ponderar a paralelização do código para tirar total partido dos processadores multicore existentes.**

# Descrição do projeto

**O projeto consiste em implementar algoritmos de multiplicação de matrizes de forma a tirar partido da cache dos processadores para acelerar o processo dos cálculos matemáticos envolvidos.**

**Para tal foram implementados 3 algoritmos diferentes.**

**Um algoritmo simples, para servir como referência, e 2 outros que tiram partido da forma como o sistema operativo usa a cache para aumentar o seu desempenho, (ao evitar o constante uso da memória principal / RAM, que é consideravelmente mais lenta que a cache dos processadores).**

Algoritmos

# A1. Algoritmo simples

**O primeiro algoritmo implementado funciona da forma tradicional que se costuma usar quando se multiplicam matrizes sem recurso a sistemas computacionais. Ou seja, cada elemento da matriz final é obtido pelo produto interno entre cada linha da matriz da esquerda em conjunto com a respetiva coluna da matriz da direita.**

**Esta implementação serve apenas como comparativo para os restantes algoritmos dado que não tira partido dos princípios de localidade usados pelos sistemas operativos recentes e tenta por outro lado, caraterizar uma implementação simples, mas também um pouco *naïve*.**

# **A2. Algoritmo de multiplicação das matrizes por linhas**

**O segundo algoritmo tira partido da cache ao agrupar os cálculos das variáveis que são repetidamente usadas.**

**Ou seja, dado que os elementos da matriz da esquerda têm que ser multiplicados por todos os elementos da respetiva fila da matriz da direita, tenta-se então agrupar esses cálculos para que os elementos da esquerda estejam sempre prontos (na cache), para serem usados para cada uma das multiplicações a realizar com os elementos das linhas da matriz da direita.**

**Isto foi implementado pegando em cada elemento da matriz da esquerda e multiplica-lo pelas linhas respetivas da matriz da direita, somando os resultados parciais nos elementos correspondentes na matriz final.**

# **A3. Algoritmo de multiplicação usando blocos**

**Este terceiro algoritmo é uma refinação do anterior, dado que para matrizes de maior dimensão, o uso de apenas um elemento da matriz da esquerda por linha da matriz da direita, desperdiça a localidade dos dados da matriz da esquerda.**

**Dado que o sistema coloca sempre dados para a cache em bloco e não um de cada vez, uma forma de aproveitar tal facto é processar seções de dados agrupados em ambas as matrizes**

**Como tal, a matriz principal foi subdividida em blocos mais pequenos de forma a agrupar a computação dos dados e rentabilizar os dados que o sistema operativo automaticamente coloca na cache.**

**Após essa agrupação por blocos, é aplicado um algoritmo semelhante ao anterior em cada uma dessas submatrizes.**

Resultados

# **Medições de tempo**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tempos de execução (segundos) | | | | | | | | | |
| Algoritmo | 600 | 1000 | 1400 | 1800 | 2200 | 2600 | 3000 | Complexidade |
| Simples | 1.13537 | 7.83 | 21.8059 | 47.4859 | 89.7186 | 169.856 | 287.637 | 2\*n^3 |
| Por linha | 0.491302 | 2.3026 | 6.41935 | 13.6234 | 24.8088 | 41.3855 | 63.3101 |
| Blocos de 16 | 0.471722 | 2.20033 | 6.0539 | 12.8937 | 23.0492 | 39.1633 | 59.9282 |
| Blocos de 32 | 0.4667 | 2.20708 | 6.11502 | 12.897 | 23.5001 | 38.9365 | 59.8617 |
| Blocos de 64 | 0.4704437 | 2.2066 | 6.08121 | 12.9498 | 23.4644 | 38.9282 | 60.1345 |
| Blocos de 128 | 0.471841 | 2.20155 | 6.12964 | 13.2095 | 23.8462 | 39.6362 | 60.1566 |
| Blocos de 256 | 0.488362 | 2.25061 | 6.30796 | 12.8911 | 23.4911 | 38.7937 | 59.5919 |

T1. Tabela com os tempos de execução de acordo com o algoritmo e tamanho da matriz usada

G1. Gráfico com o resumo dos dados da tabela T1

# Cálculo de desempenho em MFlop/s

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Desempenho (Mflop/s) | | | | | | | | |
| Algoritmo | 600 | 1000 | 1400 | 1800 | 2200 | 2600 | 3000 | Complexidade |
| Simples | 380.4927 | 255.4278 | 251.6750 | 245.6308 | 237.3644 | 206.9518 | 187.7366 | 2\*n^3 |
| Por linha | 879.2962 | 868.5833 | 854.9152 | 856.1739 | 858.4051 | 849.3796 | 852.9445 |
| Blocos de 16 | 915.7936 | 908.9546 | 906.5231 | 904.6278 | 923.9366 | 897.5750 | 901.0783 |
| Blocos de 32 | 925.6482 | 906.1747 | 897.4623 | 904.3964 | 906.2089 | 902.8033 | 902.0793 |
| Blocos de 64 | 918.2820 | 906.3718 | 902.4520 | 900.7089 | 907.5877 | 902.9958 | 897.9870 |
| Blocos de 128 | 915.5627 | 908.4509 | 895.3217 | 883.0009 | 893.0563 | 886.8660 | 897.6571 |
| Blocos de 256 | 884.5897 | 888.6480 | 870.0119 | 904.8103 | 906.5561 | 906.1265 | 906.1634 |

T2. Tabela com a conversão dos tempos medidos para MFlop/s

G2. Gráfico com o resumo da informação da tabela de desempenho em MFlop/s

# **Análise**

**Pela análise das tabelas e dos gráficos, consegue-se imediatamente ver que a implementação simples é a que tem pior desempenho e que está muito distante dos dois restantes algoritmos.**

**Tal diferença nota-se particularmente à medida que a dimensão das matrizes aumenta, dado que quanto forem as matrizes, maior será a impacto das falhas de cache.**

**Pelo gráfico de MFlop/s, consegue-se ver que a 2 e 3 implementação conseguem ser cerca de 3 a 4 vezes melhores do que a primeira.**

**A terceira implementação, apesar de ser a melhor das 3, só se revelou vantajosa em relação à segunda para matrizes de grande dimensão, e mesmo com vários tamanhos de blocos usados, tal diferença não foi muito significativa.**

**Há que referir também, que este problema de multiplicação tem um comportamento exponencial em relação ao tempo de computação, consoante se aumenta o tamanho das matrizes.**

Conclusões

**Pelos resultados obtidos nos algoritmos anteriores, consegue-se concluir que uma boa implementação paralela tem que estar sempre aliada a uma robusta implementação das seções não paralelizáveis, de forma a aproveitar todo o potencial do sistema que se está a usar.**

**Caso contrário, até uma implementação não paralelizada pode ter melhor desempenho do que uma paralelizada.**

**Este estudo vem reforçara ia deia de que para fazer sistemas eficientes, as implementações têm que ter em conta as plataformas na qual vão correr, e que que as abstrações dadas pelos sistemas operativos e *frameworks* devem ser estuadas de perto antes de serem usadas.**

Interface

# Estrutura

**Para facilitar o teste das matrizes, a implementação aceita uma forma *default* de inicialização das matrizes, no qual a matriz da esquerda é a matriz identidade e a da direita é uma matriz com as linhas com o mesmo número e estando esse número a aumentar de uma unidade entre linhas (começando em 1).**

**Esta forma de inicialização permite testar se a implementação está correta dado que o resultado final em todas as células é igual e é facilmente calculável pela suma de potências. Desta forma é possível testar se matrizes gigantes estão a ser calculadas corretamente.**

**Por outro lado permite também a importação de matrizes a partir de ficheiros, quer as que serão usadas para calcular (matriz esquerda e direita), quer a matriz com os resultados finais, de forma a confirmar automaticamente com a matriz que foi calculada. As matrizes a importar devem ter os números separados por um espaço, sendo que as matrizes esquerda e direita devem ter na primeira linha do ficheiro o número de colunas e linhas da matriz em questão (a matriz final não precisa desta linha inicial).**

**Para uma depuração mais apurada, permite também a exportação das matrizes que foram usadas nos cálculos (matrizes esquerda, direita e final).**

**Para além disso, a implementação suporta matrizes e blocos não quadrados, para poder ser usada com qualquer matriz e para poder-se melhorar o desempenho do terceiro algoritmo (ao permitir blocos retangulares).**

**Por fim, a implementação foi desenhada para triar partido dos timers de performance quer do Windows quer do Linux, e como tal, é multiplataforma.**

# **Forma de uso**

**A aplicação começa por pedir para se escolher o algoritmo a usar para a multiplicação das matrizes.**

**Depois pergunta se o utilizador quer carregar matrizes a partir de ficheiros ou se quer usar o inicializar por *default*.**

**De seguida são pedidos os tamanhos das matrizes, tendo em conta que o número de colunas da matriz da esquerda tem que ser igual ao número de linhas da matriz da direita (como tal, apenas 3 tamanhos são pedidos).**

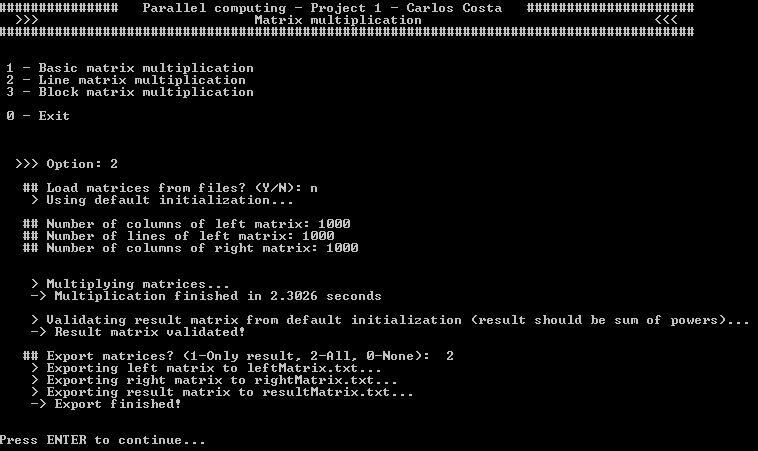
**Depois caso seja o algoritmo de blocos é pedido o número de colunas e linhas a dar aos blocos.**

**Estando a configuração feita, passa-se ao cálculo da matriz final, findo o qual é apresentado quanto essa computação demorou e mostrando se a matriz foi validade ou não (ou pela suma das potências para a inicialização *default*, ou pela comparação com o ficheiro dado com a matriz com os resultados finais).**

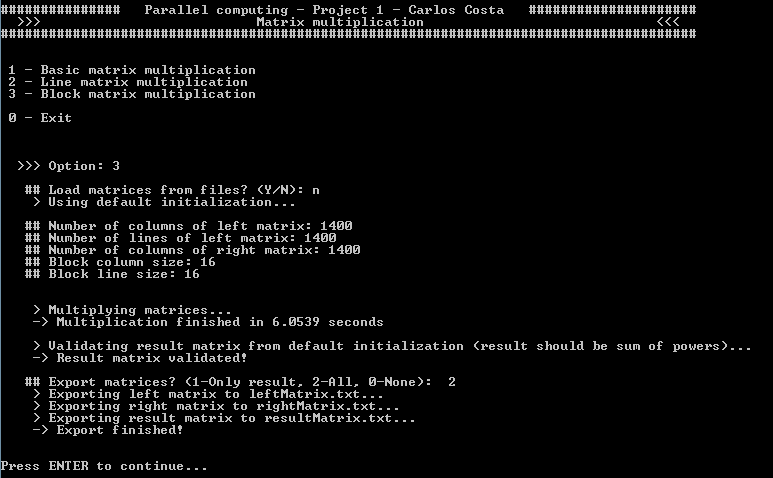
**Por fim, é perguntado se o utilizador quer exportar apenas a matriz resultado, ou também as matrizes esquerda e direita, ou então se não pretende exportar nenhuma.**

**Acabada a computação das matrizes em questão, é apresentado novamente o menu principal.**

# ***U:\Carlos Costa\Escola\MIEIC 2012-2013\2º semestre\Computação Paralela\Projetos\Projeto 1 - Multiplicação de matrizes\Entrega\screenshots\AlgSimples.PNGScreenshots***

A1. Interface para o algoritmo simples

A2. Interface para o algoritmo por linhas

A3. Interface para o algoritmo por blocos