Performance evaluation of a single core

Computação Paralela e Distribuída

2022/2023 - 2º semestre

Trabalho realizado por:

António Santos, up201907156

José Silva , up201906576

Descrição do problema

O objetivo deste projeto é estudar o desempenho do processador e da hierarquia de memória ao acessar a um grande volume de dados. Uma matriz é um conjunto retangular de números organizados por linhas e colunas, nos quais operações podem ser realizadas. Neste trabalho, realizamos o produto entre duas matrizes para acessar um grande volume de dados, permitindo ter um grande volume de números armazenados em memória. É importante destacar que neste projeto foram utilizadas duas linguagens de programação: C++ e Java.

Explicação de algoritmos

A performance relevante é o resultado da execução real de um algoritmo, logo neste projeto foram implementados três algoritmos distintos para a realização do produto entre matrizes de forma a poder fazer uma análise da performance. O produto das matrizes é obtido pela soma dos produtos dos elementos correspondentes da linha I da primeira matriz pelos elementos da coluna J da segunda matriz.

Multiplicação Simples de Matrizes

Este primeiro algoritmo é o mais simples, uma vez que não se tem atenção ao número de vezes que acede à memória, pois multiplica-se sequencialmente uma linha da primeira matriz por uma coluna da segunda matriz. Este algoritmo contém 3 ciclos e para cada iteração de um ciclo o número de passagens é o tamanho da matriz n. A complexidade deste algoritmo é O(n^3) sendo o número de acessos a memória n^3.

```
for (i = 0; i < m_ar; i++)
{
    for (j = 0; j < m_br; j++)
    {
        for (k = 0; k < m_ar; k++)
        {
            phc[i * m_ar + j] += pha[i * m_ar + k] * phb[k * m_br + j];
        }
    }
}
```

Multiplicação por linha de Matrizes

Neste algoritmo para se realizar a multiplicação das matrizes opta-se por uma abordagem mais eficiente, para isso, multiplica-se um elemento da primeira matriz pela linha correspondente, poupando desta forma acessos à memória.

```
for (i = 0; i < m_ar; i++)
{
    for (k = 0; k < m_br; k++)
    {
        for (j = 0; j < m_ar; j++)
        {
            phc[i * m_ar + j] += pha[i * m_ar + k] * phb[k * m_br + j];
        }
    }
}
```

Multiplicação por blocos

Neste algoritmo dividimos ambas as matrizes A e B em blocos de tamanho Block Size e cada bloco irá operar com o outro bloco a multiplicação de matrizes. O algoritmo não irá operar com todas as linhas e colunas , irá operar por submatrizes melhorando assim a sua performance.

```
for (i = 0; i < m_ar; i += subSize)
{
    for (k = 0; k < m_br; k += subSize)
    {
        for (j = 0; j < m_ar; j += subSize)
        {
            for (int a = i; a < calcMin( a: i,  b: subSize); a++)
            {
                for (int b = k; b < calcMin( a: k,  b: subSize); b++)
                {
                    for (int c = j; c < calcMin( a: j,  b: subSize); c++)
                    {
                        phc[a * m_ar + c] += pha[a * m_ar + b] * phb[b * m_br + c];
                    }
                }
            }
        }
    }
}
```

Métricas de desempenho

De forma a analisar o desempenho dos diferentes algoritmos foi necessário definir os parâmetros a serem medidos, para isso baseamo-nos em três diferentes parâmetros. O primeiro foi o tempo de execução da multiplicação, uma vez que para matrizes do mesmo tamanho comparando o tempo de execução é possível deduzir qual o algoritmo mais eficaz.

De seguida utilizou-se a biblioteca papi, Performance Application Programming Interface, e para isso seguimos dois eventos , o level 1 data cache misses, isto é, quando se quer aceder à memória RAM de nível um e o valor que queremos não se encontra disponível, o outro foi o level 2 data cache misses, que conta o número de vezes em que se quer aceder à memória RAM de nível 2 e o valor que queremos não se encontra disponível. Nestes dois eventos uma vez que não é possível localizar os dados é necessário fazer uma pesquisa na memória, o que aumenta o tempo de duração do pedido e consequente o tempo de duração da multiplicação das matrizes. A monitorização de eventos usando o papi apenas foi implementada para cpp e foram feitos testes para estas métricas de diferentes tamanhos de matrizes.

Resultados obtidos e análise

De forma a avaliar a performance do algoritmo básico e multiplicação em linha estes foram implementados na linguagem java e de seguida foi testado para matrizes de tamanhos de 600*600 até 3000*3000 com incrementos na dimensão de 400. Podemos ver os tempos obtidos no gráfico seguinte.

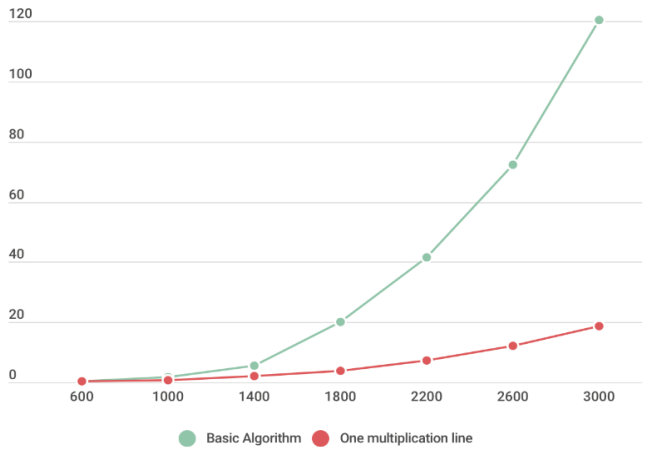


Fig 1: Tempo de execução do Java

Analisando o gráfico anterior podemos então verificar que o algoritmo de multiplicação por linha é bastante mais eficiente, sendo a sua execução bastante mais rápida.

Para a linguagem cpp foram também implementados os mesmo algoritmos para as mesmas dimensões e o resultado evidencia que a multiplicação por linha realmente é mais eficiente, como se pode ver pelo gráfico seguinte.

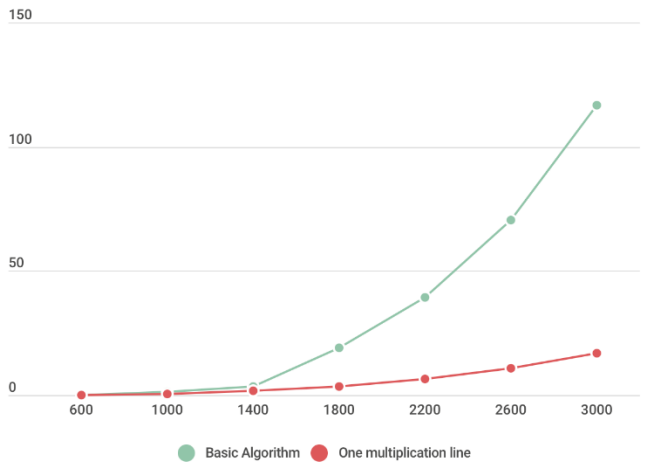


Fig 2:Tempo de Execucão C++

Concluímos que a execução em cpp é mais eficiente, uma vez que o tempo de execução é geralmente menor que em java, principalmente quando se usa um algoritmo mais eficiente para o cálculo do produto das matrizes.

Agora de maneira a testar a execução e a eficácia da multiplicação por blocos foram corridos testes para dimensões de matriz que vai de 4096 até 10240 com intervalos de 2048, em que o tamanho dos blocos também foi alterado para cada dimensão de matriz, que foi desde 128 até 1024, dobrando o tamanho dos blocos até chegar a 1024.

Uma vez que os testes foram de dimensões de 4096 até 10240, para se ter termo de comparação foi também testado o algoritmo de multiplicação por linha para essas dimensões.

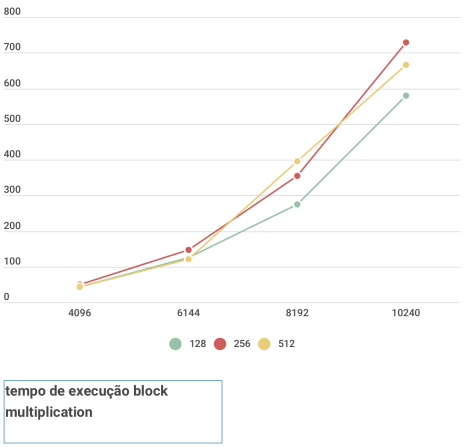
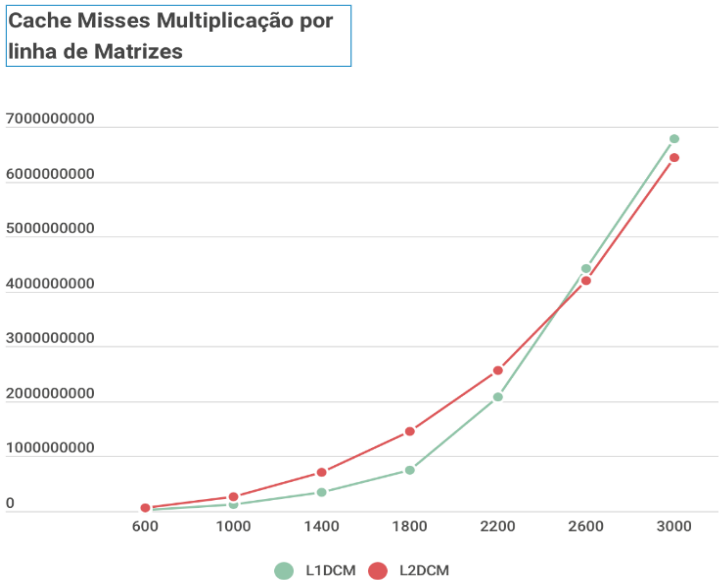
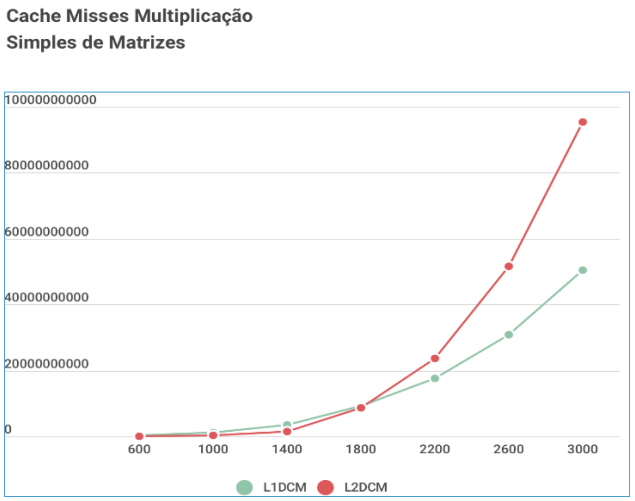


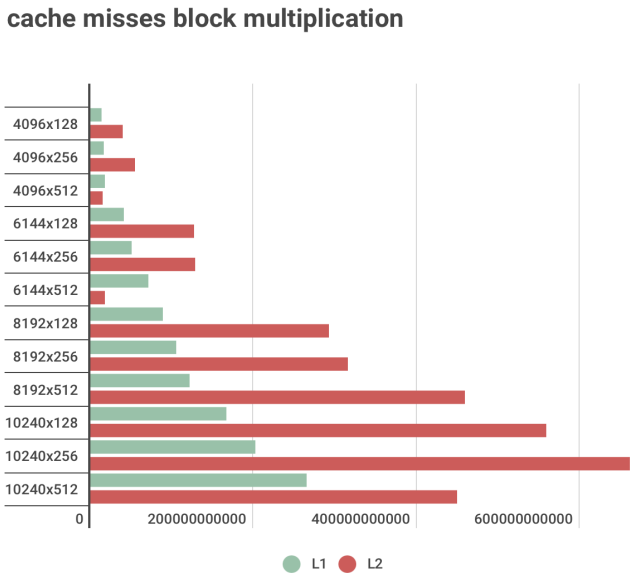
Fig 3: Tempo de execução C++ Block Multiplication

Analisando o número de data cache misses de L1 e L2, podemos ver que quanto mais cache misses maior o tempo de execução da multiplicação, pelo que algoritmos mais eficientes traduzem-se em menos data cache misses, uma vez que evitamos acessos a memória excessivos.

Cache misses



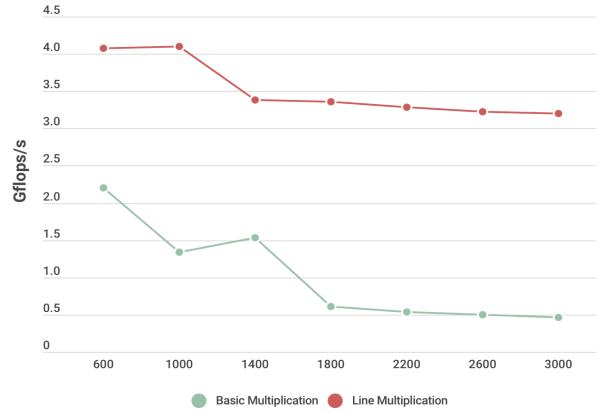




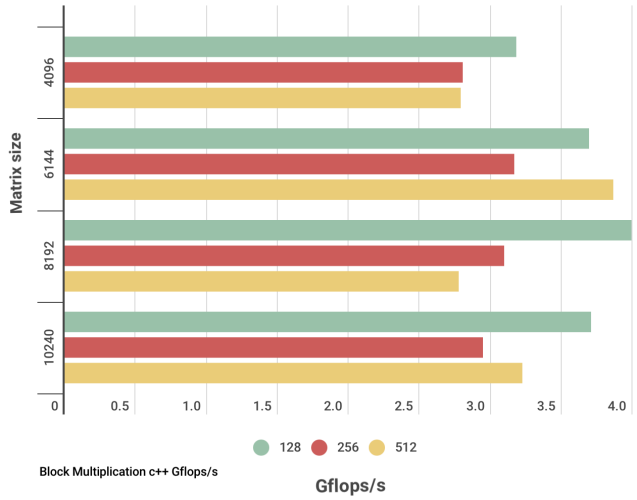
Comparação de operações de ponto flutuante

O desempenho em gigaflops varia com os valores das matrizes usadas em uma operação de cálculo, pois a intensidade computacional depende da quantidade de operações de ponto flutuante realizadas em relação ao número total de operações. Matrizes com valores grandes aumentam a intensidade computacional e resultam em um desempenho em gigaflops maior, enquanto matrizes com valores pequenos reduzem a intensidade computacional e resultam em um desempenho em gigaflops menor.

No entanto, é importante destacar que o desempenho em gigaflops não é o único fator importante que influencia o desempenho de um sistema em operações de cálculo. Outros fatores, como a arquitetura do sistema, a memória cache e a eficiência do algoritmo utilizado também podem afetar o desempenho em gigaflops.



Basic Multiplication vs Line Multiplication c++ Gflops/s



Block Multiplication c++ Gflops/s

Conclusão

Com a realização deste trabalho foi possível pôr em prática o que aprendemos nas aulas teóricas e verificar os efeitos da gestão de memória e como a implementação de algoritmos mais eficientes tem um grande peso no tempo de execução dos programas. Foi certamente um bom projeto para avaliar de que forma podemos melhorar os nossos programas.

Arquitetura computador:

Model Name: INTEL(R) CORE(TM) i7-9700 CPU @ 3.00GHz

Caches (sum of All): L1d: 256 KiB(8 instances);

L1i: 256 KiB (8 instances)

L2: 2MiB(8 instances)

L3: 12 MiB(1 instance)