

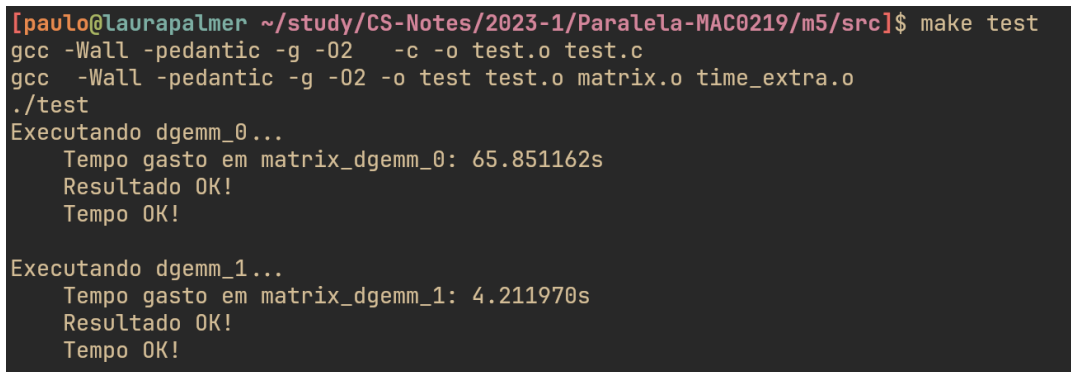
Mini Ep 5 - Otimizando o uso de cache - parte 1

Paulo Henrique Albuquerque, NUSP:12542251

2023-04-27

1 Diferença dos tempos dos algoritmos

O algoritmo otimizado apresentou melhora do tempo de execução. Em particular, o programa teste dá resultado positivo:



```
[paulo@laurapalmer ~/study/CS-Notes/2023-1/Paralela-MAC0219/m5/src]$ make test
gcc -Wall -pedantic -g -O2 -c -o test.o test.c
gcc -Wall -pedantic -g -O2 -o test test.o matrix.o time_extra.o
./test
Executando dgemm_0...
    Tempo gasto em matrix_dgemm_0: 65.851162s
    Resultado OK!
    Tempo OK!

Executando dgemm_1...
    Tempo gasto em matrix_dgemm_1: 4.211970s
    Resultado OK!
    Tempo OK!
```

Figure 1: teste.png

Para compararmos os tempos de execução, executamos os dois algoritmos para vários tamanhos N das matrizes. Para cada instância, foi rodado cada algoritmo 5 vezes e tirado a média simples dos tempos de execução. O tempo foi medido usando o programa main fornecido e o script para realizar a média foi o seguinte:

Os resultados são mostrados na figura abaixo.

A tabela abaixo resume os resultados.

Como podemos ver pelos resultados acima, é nitida a melhora no tempo de execução: o algoritmo otimizado foi mais rápido que o algoritmo simples cerca de 4 vezes nos testes.

1.1 Explicação do algoritmo otimizado

O algoritmo otimizado consiste em alterar a ordem dos laços do algoritmo simples a fim de se aproveitar da memória cache para tornar os acessos às entradas das matrizes mais rápidos. No algoritmo simples, a ordem os laços é, enquanto no algoritmo otimizado é,

A correção do algoritmo é simples e não será feita com detalhes aqui. Basicamente, o algoritmo percorre as linhas da matriz A e da matriz B . Isso é feito porque quando acessamos um elemento de uma matriz na memória, outros elementos da mesma linha são trazidos para o cache. Então, como no algoritmo de multiplicação de matrizes simples percorremos as colunas de B , trazer outros elementos da mesma linha de B é inútil e obtemos vários caches miss. No algoritmo otimizado, isso é mitigado pois percorremos linhas de B . Além disso, note que o elemento de A que será utilizado dentro do laço mais interno é armazenado em um variável temporária. Dessa forma, não precisamos acessar a memória toda vez que fizermos a atualização $C(i, j) += \dots$

2 Diferentes Maquinas

```
#!/bin/bash

NUM_EXE=5

for N in $(seq 1000 500 3000)
do

soma0=0
soma1=0

for i in $(seq $NUM_EXE)
do
    tempo0=$(./main --matrix-size $N --algorithm 0)
    tempo1=$(./main --matrix-size $N --algorithm 1)

    soma0=$(echo "$soma0 + $tempo0" | bc)
    soma1=$(echo "$soma1 + $tempo1" | bc)
done

media0=$(echo "scale=6;$soma0 / $NUM_EXE" | bc)
media1=$(echo "scale=6;$soma1 / $NUM_EXE" | bc)
echo "T(algoritmo 0, $N) = $media0."
echo "T(algoritmo 1, $N) = $media1."
echo "Razão T1/T0 = $(echo "scale=3;$media1/$media0" | bc)"
done
```

Figure 2: script.png

```
[paulo@laurapalmer ~/study/CS-Notes/2023-1/Paralela-MAC0219/m5/src]$ ./tempo
T(algoritmo 0, 1000) = .996652.
T(algoritmo 1, 1000) = .450700.
Razão T1/T0 = .452
T(algoritmo 0, 1500) = 4.068476.
T(algoritmo 1, 1500) = 1.676197.
Razão T1/T0 = .411
T(algoritmo 0, 2000) = 11.434556.
T(algoritmo 1, 2000) = 4.720929.
Razão T1/T0 = .412
T(algoritmo 0, 2500) = 21.285793.
T(algoritmo 1, 2500) = 7.856427.
Razão T1/T0 = .369
T(algoritmo 0, 3000) = 34.252814.
T(algoritmo 1, 3000) = 13.565845.
Razão T1/T0 = .396
```

Figure 3: resultados.png

```

for (i = 0; i < n; ++i)
  for (j = 0; j < n; ++j)
  {
    double sum = 0;
    for (k = 0; k < n; ++k)
      sum += A(i, k)*B(k, j);
    C(i, j) = sum;
  }

```

Figure 4: simples.png

```

for (i = 0; i < n; ++i)
  for (k = 0; k < n; ++k)
  {
    double tmp = A(i,k);
    for (j = 0; j < n; ++j)
      C(i,j) += tmp * B(k,j);
  }

```

Figure 5: otimizado.png