Danilo Yui Honda - 10419625

Computação Paralela

Multiplicação de Matrizes – Acesso à Memória e Otimizações

---

1 - Implementações

Neste trabalho prático de Computação Paralela, implementei três variações da multiplicação de matrizes em C++:

- **Acesso por linha (row order)**
- **Acesso por coluna (column order)**
- **Acesso por blocagem (block order)**

O objetivo foi analisar o desempenho e o impacto no uso da memória cache utilizando diferentes padrões de acesso, além de avaliar os efeitos de otimizações de compilador na performance.

---

2 - Compilações e Otimizações

Cada versão foi compilada com e sem otimizações utilizando a flag `-O3` do compilador g++. As otimizações promovem reordenação de instruções, unrolling de loops e melhor uso dos registradores, o que impacta diretamente no tempo de execução.

---

3 - Análise de Desempenho

Abaixo estão os tempos médios de execução registrados para cada variação:

Ordem de Acesso	Tempo sem Otimização	Tempo com Otimização (-O3)
Row Order	1.025.128 µs	100.514 µs
Column Order	998.418 µs	102.119 µs
Block Order	1.146.342 µs	65.790 µs

Observa-se que:

- Com otimização, os tempos caem drasticamente.
- O acesso **block order**, apesar de ser o mais lento sem otimização, se torna o mais rápido com `-O3`.
- A diferença de desempenho entre **row** e **column** diminui com otimizações.

---

4 - Análise com Valgrind (Cachegrind)

Para entender o impacto no uso de cache, utilizei o `valgrind --tool=cachegrind`. Os principais dados estão abaixo:

Row Order:

- Instruções: 306.349.275
- D1 Misses: 3.537.461
- LL Misses: 63.113
- D1 Miss Rate: 2.2%
- LL Miss Rate: 0.0%

Column Order:

- Instruções: 818.349.855
- D1 Misses: 9.418.806
- LL Misses: 63.113
- D1 Miss Rate: 2.2%
- LL Miss Rate: 0.0%

Block Order:

- Instruções: 227.822.044
- D1 Misses: 92.190
- LL Misses: 63.117
- D1 Miss Rate: 0.1%
- LL Miss Rate: 0.0%

Observações:

- O **acesso por coluna** gera mais referências de instruções e maior número de *misses* em cache de dados (D1).
- **A blocagem** apresenta o menor número de *misses* em D1, evidenciando seu benefício na localidade de dados.
- Apesar de ligeiramente mais instruções, o **row order** ainda tem desempenho superior ao column, principalmente com otimização.

5 - Conclusão

O trabalho demonstrou como padrões de acesso à memória e otimizações de compilador impactam significativamente no desempenho de algoritmos como a multiplicação de matrizes.

- A versão **block order** é mais eficiente em uso de cache e, com otimização, foi a mais rápida.
- O uso da flag `-O3` traz ganhos expressivos, mostrando como compiladores modernos conseguem explorar melhor o hardware.
- Ferramentas como o Valgrind são essenciais para entender gargalos e padrões de acesso à memória, além de ajudar na escolha de estratégias mais eficientes.