Threads em C++

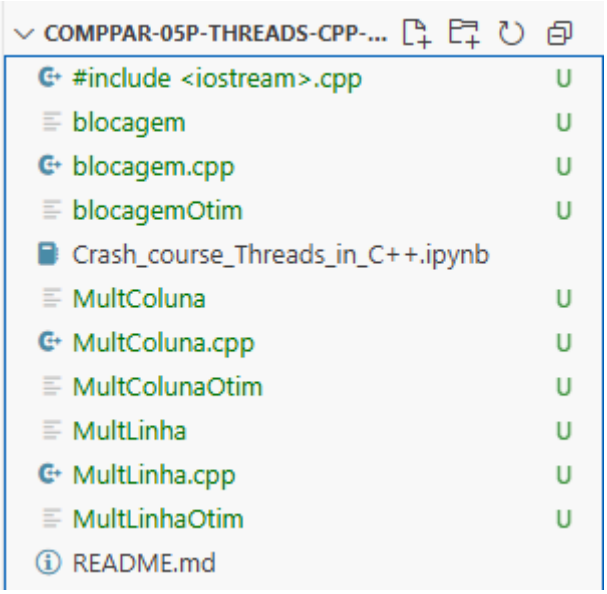
Nome: Vitor Tibães Santos

RA: 10418976

Turma: 05P

Criando executáveis:

```
@Moutt ➜/workspaces/comppar-05p-threads-cpp-Moutt (main) $ g++ -O0 MultLinha.cpp -o MultLinha
@Moutt ➜/workspaces/comppar-05p-threads-cpp-Moutt (main) $ g++ -O0 MultColuna.cpp -o MultColuna
@Moutt ➜/workspaces/comppar-05p-threads-cpp-Moutt (main) $ g++ -O0 blocagem.cpp -o blocagem
@Moutt ➜/workspaces/comppar-05p-threads-cpp-Moutt (main) $ g++ -O3 MultLinha.cpp -o MultLinhaOtim
@Moutt ➜/workspaces/comppar-05p-threads-cpp-Moutt (main) $ g++ -O3 MultColuna.cpp -o MultColunaOtim
@Moutt ➜/workspaces/comppar-05p-threads-cpp-Moutt (main) $ g++ -O3 blocagem.cpp -o blocagemOtim
```



Executando arquivos:

```
@Moutt →/workspaces/comppar-05p-threads-cpp-Moutt (main) $ ./MultColuna
Tempo (coluna por coluna): 1.5873 segundos
@Moutt →/workspaces/comppar-05p-threads-cpp-Moutt (main) $ ./MultLinha
Tempo (linha por linha): 1.72689 segundos
@Moutt →/workspaces/comppar-05p-threads-cpp-Moutt (main) $ ./blocagem
Tempo (com blocagem): 1.90947 segundos
@Moutt →/workspaces/comppar-05p-threads-cpp-Moutt (main) $ ./MultLinhaOtim
Tempo (linha por linha): 0.127728 segundos
@Moutt →/workspaces/comppar-05p-threads-cpp-Moutt (main) $ ./MultColunaOtim
Tempo (coluna por coluna): 0.132301 segundos
@Moutt →/workspaces/comppar-05p-threads-cpp-Moutt (main) $ ./blocagemOtim
Tempo (com blocagem): 0.0822395 segundos
```

	MultLinha	MultiColuna	blocagem
Não Otimizado	1.72689 segs	1.5873 segs	1.90947 segs
Otimizado	0.127728 segs	0.132301 segs	0.0822395 segs

Valgrind

MultLinha

```
Tempo (com blocagem): 0.0822395 segundos
@Moutt →/workspaces/comppar-05p-threads-cpp-Moutt (main) $ valgrind --tool=cachegrind ./MultLinha
==16426== Cachegrind, a cache and branch-prediction profiler
==16426== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==16426== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==16426== Command: ./MultLinha
==16426==
--16426-- warning: L3 cache found, using its data for the LL simulation.
Tempo (linha por linha): 54.6081 segundos
==16426==
==16426== I   refs:      15,828,200,102
==16426== I1  misses:             2,276
==16426== LLi misses:             2,088
==16426== I1  miss rate:           0.00%
==16426== LLi miss rate:           0.00%
==16426==
==16426== D   refs:       9,158,672,281  (5,772,331,266 rd   + 3,386,341,015 wr)
==16426== D1  misses:       188,021,526  (  187,829,963 rd   +       191,563 wr)
==16426== LLd misses:           104,675  (        8,381 rd   +        96,294 wr)
==16426== D1  miss rate:            2.1% (          3.3%     +           0.0%  )
==16426== LLd miss rate:            0.0% (          0.0%     +           0.0%  )
==16426==
==16426== LL refs:          188,023,802  (  187,832,239 rd   +       191,563 wr)
==16426== LL misses:            106,763  (       10,469 rd   +        96,294 wr)
==16426== LL miss rate:             0.0% (          0.0%     +           0.0%  )
```

MultColuna

```
@Moutt ➜/workspaces/comppar-05p-threads-cpp-Moutt (main) $ valgrind --tool=cachegrind ./MultColuna
==17065== Cachegrind, a cache and branch-prediction profiler
==17065== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==17065== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==17065== Command: ./MultColuna
==17065==
--17065-- warning: L3 cache found, using its data for the LL simulation.
Tempo (coluna por coluna): 54.8968 segundos
==17065==
==17065== I   refs:      15,828,200,225
==17065== I1  misses:             2,275
==17065== LLi misses:             2,087
==17065== I1  miss rate:           0.00%
==17065== LLi miss rate:           0.00%
==17065==
==17065== D   refs:       9,158,672,328  (5,772,331,296 rd   + 3,386,341,032 wr)
==17065== D1  misses:       188,303,461  (  187,893,275 rd   +       410,186 wr)
==17065== LLd misses:           104,674  (        8,380 rd   +        96,294 wr)
==17065== D1  miss rate:            2.1% (          3.3%     +           0.0%  )
==17065== LLd miss rate:            0.0% (          0.0%     +           0.0%  )
==17065==
==17065== LL refs:          188,305,736  (  187,895,550 rd   +       410,186 wr)
==17065== LL misses:            106,761  (       10,467 rd   +        96,294 wr)
==17065== LL miss rate:             0.0% (          0.0%     +           0.0%  )
```

Blocagem:

```
@Moutt ➜/workspaces/comppar-05p-threads-cpp-Moutt (main) $ valgrind --tool=cachegrind ./blocagem
==17928== Cachegrind, a cache and branch-prediction profiler
==17928== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==17928== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==17928== Command: ./blocagem
==17928==
--17928-- warning: L3 cache found, using its data for the LL simulation.
Tempo (com blocagem): 69.1834 segundos
==17928==
==17928== I   refs:      19,219,481,285
==17928== I1  misses:             2,281
==17928== LLi misses:             2,098
==17928== I1  miss rate:           0.00%
==17928== LLi miss rate:           0.00%
==17928==
==17928== D   refs:      11,116,440,701  (6,949,443,079 rd   + 4,166,997,622 wr)
==17928== D1  misses:         1,719,164  (    1,558,977 rd   +       160,187 wr)
==17928== LLd misses:           104,673  (        8,379 rd   +        96,294 wr)
==17928== D1  miss rate:            0.0% (          0.0%     +           0.0%  )
==17928== LLd miss rate:            0.0% (          0.0%     +           0.0%  )
==17928==
==17928== LL refs:            1,721,445  (    1,561,258 rd   +       160,187 wr)
==17928== LL misses:            106,771  (       10,477 rd   +        96,294 wr)
==17928== LL miss rate:             0.0% (          0.0%     +           0.0%  )
@Moutt ➜/workspaces/comppar-05p-threads-cpp-Moutt (main) $
```

Analise

A blocagem teve drástica redução nos misses de cache D1: passou de 188M para 1.7M misses!

O acesso por colunas teve um número de misses de cache um pouco maior que o acesso por linhas.

O número de misses de cache L1 (D1) no acesso por colunas é um pouco pior do que no acesso por linhas, confirmando que percorrer colunas prejudica o cache.

Blocagem teve o menor índice de misses no cache L1, o que confirma sua eficiência em reduzir falhas de cache.

Acessar a matriz por colunas é pior para o cache do que acessar por linhas, pois a memória é armazenada em ordem de linha.

Blocagem reduziu drasticamente as falhas no cache L1, mas não melhorou o tempo de execução.

Testar diferentes tamanhos de blocos pode melhorar a eficiência da blocagem.

Compilar com otimização (-O3) pode ajudar a melhorar o desempenho da blocagem.