

Universidade Presbiteriana Mackenzie  
Ciência da Computação – 05P11  
Computação Paralela  
20/02/2025

Alan Meniuk Gleizer  
RA 10416804

## Atividade – Monitoramento de Processos

**1. Crie dois executáveis para a parte de multiplicação de matrizes: um que percorre em ordem de linha e outro que percorre em ordem de coluna.**

Código fonte nos arquivos:

mult\_mat\_linhas.cpp  
mult\_mat\_colunas.cpp

Neste mesmo repositório.

**2. Crie um terceiro executável para utilizar corretamente o cache (hierarquia de memória).**

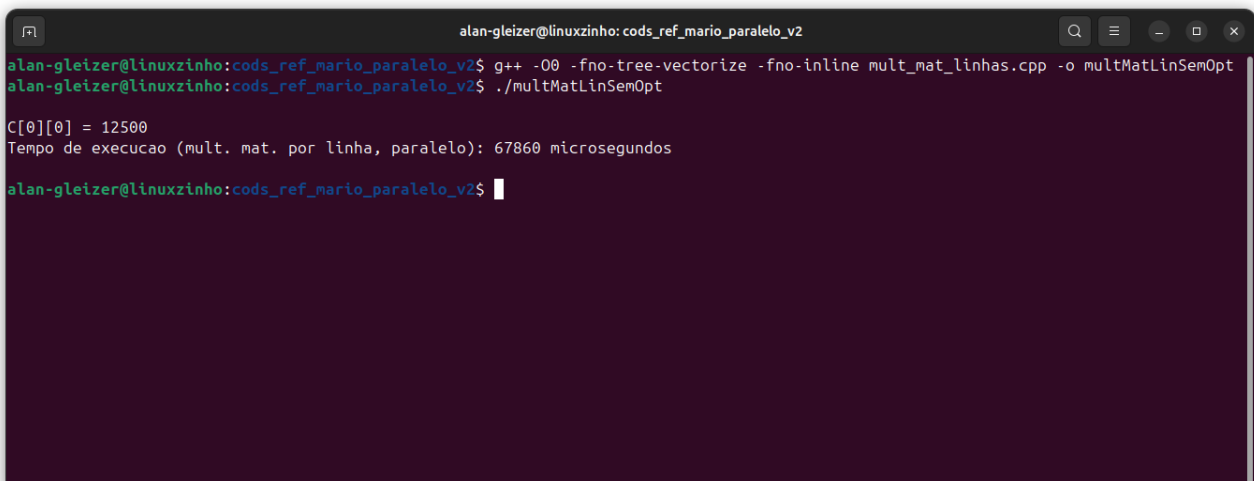
Código fonte no arquivo:

mult\_mat\_blocagem.cpp

Neste mesmo repositório.

**Para cada opção, você deve medir o tempo de execução:**

**a) desligando todas as otimizações do compilador.**



```
alan-gleizer@linuxzinho: cods_ref_mario_paralelo_v2
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ g++ -O0 -fno-tree-vectorize -fno-inline mult_mat_linhas.cpp -o multMatLinSemOpt
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ ./multMatLinSemOpt

C[0][0] = 12500
Tempo de execucao (mult. mat. por linha, paralelo): 67860 microsegundos

alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$
```

```
alan-gleizer@linuxzinho: cods_ref_mario_paralelo_v2
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ g++ -O0 -fno-tree-vectorize -fno-inline mult_mat_colunas.cpp -o multMatColSemOpt
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ ./multMatColSemOpt

C[0][0] = 12500
Tempo de execucao (mult. mat. por coluna, paralelo): 73523 microsegundos

alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$
```

```
alan-gleizer@linuxzinho: cods_ref_mario_paralelo_v2
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ g++ -O0 -fno-tree-vectorize -fno-inline mult_mat_blocagem.cpp -o multMatBlocSemOpt
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ ./multMatBlocSemOpt

C[0][0] = 12500
Tempo de execucao (blocagem, paralelo): 62394 microsegundos

alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$
```

## b) ligando a otimização máxima.

```
alan-gleizer@linuxzinho: cods_ref_mario_paralelo_v2
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ g++ -Ofast -march=native mult_mat_linhas.cpp -o multMatLinComOpt
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ ./multMatLinComOpt

C[0][0] = 12500
Tempo de execucao (mult. mat. por linha, paralelo): 10767 microsegundos

alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$
```

```
alan-gleizer@linuxzinho: cods_ref_mario_paralelo_v2
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ g++ -Ofast -march=native mult_mat_colunas.cpp -o multMatColComOpt
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ ./multMatColComOpt

C[0][0] = 12500
Tempo de execucao (mult. mat. por coluna, paralelo): 10558 microsegundos

alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$
```

```
alan-gleizer@linuxzinho: cods_ref_mario_paralelo_v2
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ g++ -Ofast -march=native mult_mat_blocagem.cpp -o multMatBlocComOpt
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ ./multMatBlocComOpt

C[0][0] = 12500
Tempo de execucao (blocagem, paralelo): 10795 microsegundos

alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$
```

### 3. Faça também uma análise do padrão de acesso ao cache de todas as versões utilizando o utilitário valgrind.

Neste ponto é importante relatar que não foi possível executar o valgrind para os executáveis compilados com as flags `-Ofast` e `-march=native`. A primeira se trata de uma flag que ativa `-O3`, o maior nível de otimização, e ainda ignora conformidade com alguns padrões de representação para ganhar ainda mais velocidade. A segunda flag, por sua vez, ativa compilação com instruções específicas da CPU para ainda mais otimização. Nestes casos, o valgrind detecta uma instrução não-reconhecida no programa e aborta a execução, como no exemplo abaixo:

```
alan-gleizer@linuxzinho: cods_ref_mario_paralelo_v2
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ valgrind --tool=cachegrind --cache-sim=yes ./multMatLinComOpt
==9542== Cachegrind, a high-precision tracing profiler
==9542== Copyright (C) 2002-2024, and GNU GPL'd, by Nicholas Nethercote et al.
==9542== Using Valgrind-3.23.0 and LibVEX; rerun with -h for copyright info
==9542== Command: ./multMatLinComOpt
==9542==
--9542-- warning: L3 cache found, using its data for the LL simulation.
vex amd64->IR: unhandled instruction bytes: 0x62 0xF2 0x7D 0x48 0x7C 0xC1 0x48 0x8D 0xB0 0xD0
vex amd64->IR:   REX=0 REX.W=0 REX.R=0 REX.X=0 REX.B=0
vex amd64->IR:   VEX=0 VEX.L=0 VEX.nVVVV=0x0 ESC=NONE
vex amd64->IR:   PFX.66=0 PFX.F2=0 PFX.F3=0
==9542== valgrind: Unrecognised instruction at address 0x10a4d8.
==9542==    at 0x10A4D8: main (in /home/alan-gleizer/GitHubRepos/ComputacaoParalela/atividade03_mult_mat_threads/cods_ref_mari
o_paralelo_v2/multMatLinComOpt)
==9542== Your program just tried to execute an instruction that Valgrind
==9542== did not recognise. There are two possible reasons for this.
==9542== 1. Your program has a bug and erroneously jumped to a non-code
==9542==    location. If you are running Memcheck and you just saw a
==9542==    warning about a bad jump, it's probably your program's fault.
==9542== 2. The instruction is legitimate but Valgrind doesn't handle it,
==9542==    i.e. it's Valgrind's fault. If you think this is the case or
==9542==    you are not sure, please let us know and we'll try to fix it.
==9542== Either way, Valgrind will now raise a SIGILL signal which will
==9542== probably kill your program.
==9542==
==9542== Process terminating with default action of signal 4 (SIGILL)
==9542== Illegal opcode at address 0x10A4D8
==9542==    at 0x10A4D8: main (in /home/alan-gleizer/GitHubRepos/ComputacaoParalela/atividade03_mult_mat_threads/cods_ref_mari
o_paralelo_v2/multMatLinComOpt)
==9542==
==9542== I refs:      1,878,134
==9542== I1 misses:    1,762
==9542== LLi misses:   1,732
==9542== I1 miss rate: 0.09%
==9542== LLi miss rate: 0.09%
==9542==
==9542== D refs:      659,456 (487,501 rd + 171,955 wr)
==9542== D1 misses:   15,447 ( 13,242 rd +  2,205 wr)
==9542== LLD misses:    8,873 (  7,494 rd +  1,379 wr)
==9542== D1 miss rate:  2.3% (  2.7% +  1.3% )
==9542== LLD miss rate: 1.3% (  1.5% +  0.8% )
==9542==
==9542== LL refs:      17,209 ( 15,004 rd +  2,205 wr)
==9542== LL misses:    10,605 (  9,226 rd +  1,379 wr)
==9542== LL miss rate:  0.4% (  0.4% +  0.8% )
Illegal instruction (core dumped)
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$
```

É provável que se trate de uma incompatibilidade do valgrind com instruções específicas da plataforma AMD Zen 4, na qual os programas foram executados. Dessa forma, foi necessário recompilar as versões otimizadas utilizando a flag -O3 e omitindo - march=native. Os resultados são apresentados abaixo:

```
alan-gleizer@linuxzinho: cods_ref_mario_paralelo_v2
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ g++ -O0 -fno-tree-vectorize -fno-inline mult_mat_linhas.cpp -o multMatLinSemOpt
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ ./multMatLinSemOpt

C[0][0] = 12500
Tempo de execucao (mult. mat. por linha, paralelo): 67895 microsegundos

alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ valgrind --tool=cachegrind --cache-sim=yes ./multMatLinSemOpt
==10685== Cachegrind, a high-precision tracing profiler
==10685== Copyright (C) 2002-2024, and GNU GPL'd, by Nicholas Nethercote et al.
==10685== Using Valgrind-3.23.0 and LibVEX; rerun with -h for copyright info
==10685== Command: ./multMatLinSemOpt
==10685==
--10685-- warning: L3 cache found, using its data for the LL simulation.

C[0][0] = 12500
Tempo de execucao (mult. mat. por linha, paralelo): 20369601 microsegundos

==10685==
==10685== I refs:          10,766,878,762
==10685== I1 misses:         2,857
==10685== L1i misses:       2,660
==10685== I1 miss rate:      0.00%
==10685== L1i miss rate:    0.00%
==10685==
==10685== D refs:          6,009,398,826 (4,005,935,156 rd + 2,003,463,670 wr)
==10685== D1 misses:       180,815,541 ( 180,514,800 rd +    300,741 wr)
==10685== L1d misses:       57,398 (    7,783 rd +    49,615 wr)
==10685== D1 miss rate:     3.0% (    4.5% +    0.0% )
==10685== L1d miss rate:    0.0% (    0.0% +    0.0% )
==10685==
==10685== LL refs:          180,818,398 ( 180,517,657 rd +    300,741 wr)
==10685== LL misses:        60,058 (    10,443 rd +    49,615 wr)
==10685== LL miss rate:     0.0% (    0.0% +    0.0% )
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$
```

```
alan-gleizer@linuxzinho: cods_ref_mario_paralelo_v2
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ g++ -O0 -fno-tree-vectorize -fno-inline mult_mat_colunas.cpp -o multMatColSemOpt
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ ./multMatColSemOpt

C[0][0] = 12500
Tempo de execucao (mult. mat. por coluna, paralelo): 69329 microsegundos

alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ valgrind --tool=cachegrind --cache-sim=yes ./multMatColSemOpt
==11540== Cachegrind, a high-precision tracing profiler
==11540== Copyright (C) 2002-2024, and GNU GPL'd, by Nicholas Nethercote et al.
==11540== Using Valgrind-3.23.0 and LibVEX; rerun with -h for copyright info
==11540== Command: ./multMatColSemOpt
==11540==
--11540-- warning: L3 cache found, using its data for the LL simulation.
--11540-- warning: specified LL cache: line_size 64 assoc 1 total_size 201,326,592
--11540-- warning: simulated LL cache: line_size 64 assoc 2 total_size 268,435,456

C[0][0] = 12500
Tempo de execucao (mult. mat. por coluna, paralelo): 25779081 microsegundos

==11540==
==11540== I refs:          10,766,878,508
==11540== I1 misses:         2,862
==11540== L1i misses:       2,614
==11540== I1 miss rate:      0.00%
==11540== L1i miss rate:    0.00%
==11540==
==11540== D refs:          6,009,398,778 (4,005,935,108 rd + 2,003,463,670 wr)
==11540== D1 misses:       180,534,676 ( 180,233,934 rd +    300,742 wr)
==11540== L1d misses:       57,191 (    7,573 rd +    49,618 wr)
==11540== D1 miss rate:     3.0% (    4.5% +    0.0% )
==11540== L1d miss rate:    0.0% (    0.0% +    0.0% )
==11540==
==11540== LL refs:          180,537,538 ( 180,236,796 rd +    300,742 wr)
==11540== LL misses:        59,805 (    10,187 rd +    49,618 wr)
==11540== LL miss rate:     0.0% (    0.0% +    0.0% )
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$
```

```

alan-gleizer@linuxzinho: cods_ref_mario_paralelo_v2
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ g++ -O0 -fno-tree-vectorize -fno-inline mult_mat_blocagem.cpp -o multMatBlocSemOpt
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ ./multMatBlocSemOpt

C[0][0] = 12500
Tempo de execucao (blocagem, paralelo): 63667 microsegundos

alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ valgrind --tool=cachegrind --cache-sim=yes ./multMatBlocSemOpt
==11152== Cachegrind, a high-precision tracing profiler
==11152== Copyright (C) 2002-2024, and GNU GPL'd, by Nicholas Nethercote et al.
==11152== Using Valgrind-3.23.0 and LibVEX; rerun with -h for copyright info
==11152== Command: ./multMatBlocSemOpt
==11152==
--11152-- warning: L3 cache found, using its data for the LL simulation.

C[0][0] = 12500
Tempo de execucao (blocagem, paralelo): 18681878 microsegundos

==11152==
==11152== I refs:          10,767,045,151
==11152== I1 misses:         2,856
==11152== L1i misses:        2,668
==11152== I1 miss rate:      0.00%
==11152== L1i miss rate:    0.00%
==11152==
==11152== D refs:          6,009,514,894 (4,006,033,464 rd + 2,003,481,430 wr)
==11152== D1 misses:        180,813,749 ( 180,513,008 rd +    300,741 wr)
==11152== L1d misses:        57,402 (    7,787 rd +    49,615 wr)
==11152== D1 miss rate:      3.0% (    4.5% +    0.0% )
==11152== L1d miss rate:    0.0% (    0.0% +    0.0% )
==11152==
==11152== LL refs:          180,816,605 ( 180,515,864 rd +    300,741 wr)
==11152== LL misses:         60,070 (    10,455 rd +    49,615 wr)
==11152== LL miss rate:      0.0% (    0.0% +    0.0% )
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$

```

```

alan-gleizer@linuxzinho: cods_ref_mario_paralelo_v2
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ ./multMatLinComOpt

C[0][0] = 12500
Tempo de execucao (mult. mat. por linha, paralelo): 13536 microsegundos

alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ valgrind --tool=cachegrind --cache-sim=yes ./multMatLinComOpt
==9910== Cachegrind, a high-precision tracing profiler
==9910== Copyright (C) 2002-2024, and GNU GPL'd, by Nicholas Nethercote et al.
==9910== Using Valgrind-3.23.0 and LibVEX; rerun with -h for copyright info
==9910== Command: ./multMatLinComOpt
==9910==
--9910-- warning: L3 cache found, using its data for the LL simulation.

C[0][0] = 12500
Tempo de execucao (mult. mat. por linha, paralelo): 2109500 microsegundos

==9910==
==9910== I refs:          1,005,492,958
==9910== I1 misses:         2,517
==9910== L1i misses:        2,383
==9910== I1 miss rate:      0.00%
==9910== L1i miss rate:    0.00%
==9910==
==9910== D refs:          376,418,097 (375,789,820 rd +  628,277 wr)
==9910== D1 misses:        180,316,399 (180,015,604 rd +  300,795 wr)
==9910== L1d misses:        57,228 (    7,624 rd +    49,604 wr)
==9910== D1 miss rate:      47.9% (    47.9% +    47.9% )
==9910== L1d miss rate:    0.0% (    0.0% +    7.9% )
==9910==
==9910== LL refs:          180,318,916 (180,018,121 rd +  300,795 wr)
==9910== LL misses:         59,611 (    10,007 rd +    49,604 wr)
==9910== LL miss rate:      0.0% (    0.0% +    7.9% )
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$

```

```
alan-gleizer@linuxzinho: cods_ref_mario_paralelo_v2
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ ./multMatColComOpt

C[0][0] = 12500
Tempo de execucao (mult. mat. por coluna, paralelo): 10737 microsegundos

alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ valgrind --tool=cachegrind --cache-sim=yes ./multMatColComOpt
==10150== Cachegrind, a high-precision tracing profiler
==10150== Copyright (C) 2002-2024, and GNU GPL'd, by Nicholas Nethercote et al.
==10150== Using Valgrind-3.23.0 and LibVEX; rerun with -h for copyright info
==10150== Command: ./multMatColComOpt
==10150==
--10150-- warning: L3 cache found, using its data for the LL simulation.
--10150-- warning: specified LL cache: line_size 64  assoc 1  total_size 201,326,592
--10150-- warning: simulated LL cache: line_size 64  assoc 2  total_size 268,435,456

C[0][0] = 12500
Tempo de execucao (mult. mat. por coluna, paralelo): 2432078 microsegundos

==10150==
==10150== I refs:      1,005,491,996
==10150== I1 misses:    2,518
==10150== L1i misses:  2,380
==10150== I1 miss rate: 0.00%
==10150== L1i miss rate: 0.00%
==10150==
==10150== D refs:      376,917,063 (376,288,799 rd + 628,264 wr)
==10150== D1 misses:  180,690,621 (180,389,832 rd + 300,789 wr)
==10150== L1d misses:  57,173 ( 7,569 rd + 49,604 wr)
==10150== D1 miss rate: 47.9% ( 47.9% + 47.9% )
==10150== L1d miss rate: 0.0% ( 0.0% + 7.9% )
==10150==
==10150== LL refs:      180,693,139 (180,392,350 rd + 300,789 wr)
==10150== LL misses:    59,553 ( 9,949 rd + 49,604 wr)
==10150== LL miss rate: 0.0% ( 0.0% + 7.9% )
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$
```

```
alan-gleizer@linuxzinho: cods_ref_mario_paralelo_v2
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ g++ -O3 mult_mat_blocagem.cpp -o multMatBlocComOpt
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ ./multMatBlocComOpt

C[0][0] = 12500
Tempo de execucao (blocagem, paralelo): 11064 microsegundos

alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$ valgrind --tool=cachegrind --cache-sim=yes ./multMatBlocComOpt
==10384== Cachegrind, a high-precision tracing profiler
==10384== Copyright (C) 2002-2024, and GNU GPL'd, by Nicholas Nethercote et al.
==10384== Using Valgrind-3.23.0 and LibVEX; rerun with -h for copyright info
==10384== Command: ./multMatBlocComOpt
==10384==
--10384-- warning: L3 cache found, using its data for the LL simulation.
--10384-- warning: specified LL cache: line_size 64  assoc 1  total_size 201,326,592
--10384-- warning: simulated LL cache: line_size 64  assoc 2  total_size 268,435,456

C[0][0] = 12500
Tempo de execucao (blocagem, paralelo): 2455294 microsegundos

==10384==
==10384== I refs:      1,005,658,910
==10384== I1 misses:    2,524
==10384== L1i misses:  2,385
==10384== I1 miss rate: 0.00%
==10384== L1i miss rate: 0.00%
==10384==
==10384== D refs:      376,500,607 (375,870,710 rd + 629,897 wr)
==10384== D1 misses:  180,355,924 (180,055,145 rd + 300,779 wr)
==10384== L1d misses:  57,173 ( 7,569 rd + 49,604 wr)
==10384== D1 miss rate: 47.9% ( 47.9% + 47.8% )
==10384== L1d miss rate: 0.0% ( 0.0% + 7.9% )
==10384==
==10384== LL refs:      180,358,448 (180,057,669 rd + 300,779 wr)
==10384== LL misses:    59,558 ( 9,954 rd + 49,604 wr)
==10384== LL miss rate: 0.0% ( 0.0% + 7.9% )
alan-gleizer@linuxzinho:cods_ref_mario_paralelo_v2$
```

## Comentários

O exercício proposto foi bastante desafiador em múltiplas frentes de aprendizado. EM primeiro lugar, se tratou do segundo contato com a linguagem C++, mas com sintaxe mais avançada do que o visto na disciplina de SO. Ainda que C++ apresente uma série de similaridades com C, as diferenças são suficientes para dificultar a legibilidade do código e o seu entendimento. Em particular é possível citar os operadores `<<` e `::`, o uso de objetos e generics com sintaxe distinta tanto de C quanto de Java. Após estudo das estruturas e particularidades de C++, foi possível enfrentar o desafio seguinte: a programação paralelizada em si.

A paralelização dos algoritmos de multiplicação de matrizes foi facilitada pelos exemplos fornecidos no Jupyter Notebook da aula. A estrutura básica das abordagens por linha, por coluna e com blocagem foi mantida, mas em todos os casos foi necessário estruturar uma nova abordagem para implementação dos códigos. Pessoalmente, essa nova abordagem me lembrou o tipo de análise necessária para resolver um problema utilizando recursividade. Foi necessário, por exemplo, entender que seria necessário passar os índices das linhas/colunas que cada thread trabalharia como parâmetros das funções, da mesma forma que utilizamos parâmetros em recursividade para iterar sobre arrays/matrizes. Da mesma forma, foi necessário desenvolver código para definir como seria feita a divisão das linhas/colunas e como armazenar essas informações para as várias interações de loop nos quais executamos as diferentes threads. No caso específico da blocagem, o processo foi ainda mais difícil.

Por fim, houve muita dificuldade no momento de compilação e análise com o Valgrind. De acordo com as pesquisas realizadas, as flags `-O0` e `-Ofast` representam os níveis mínimos e máximos de otimização no g++. Também ficou claro que é recomendado utilizar `-march=native` para aproveitar particularidades de otimização da arquitetura de cada processador específico. Foi necessário bastante pesquisa e debug para entender, ou desenvolver a hipótese, que algumas dessas flags resultam em instruções que não são suportadas pelo Valgrind.

Compilando os resultados, é possível ver uma série de padrões interessantes:



Programa		Tempo exec. (µs)	Análise Valgrind (métricas que apresentaram variabilidade)			
			Num. instruções	D1 miss rate (%)		
Ordem coluna	-O0	69329	$10,8 * 10^9$	3,0		
	-O3	10737	$1,0 * 10^9$	47,9		
	-Ofast -march=native	10558	-	-		
Ordem linha	-O0	67895	$10,8 * 10^9$	3,0		
	-O3	13536	$1,0 * 10^9$	47,9		
	-Ofast -march=native	10767	-	-		
Por blocagem	-O0	63667	$10,8 * 10^9$	3,0		
	-O3	11064	$1,0 * 10^9$	47,9		
	-Ofast -march=native	10795	-	-		

As otimizações de compilador (-O3, -Ofast) são claramente bastante eficazes, reduzindo o número de operações em aproximadamente 10 vezes e o tempo de execução em quase 6x. Conforme a pesquisa realizada sobre ILP, isso se deve ao fato de que o compilador reorganiza o código aplicando técnicas como, unrolling de loops e pré-buscas de dados. Isso faz com que o programa execute muito mais rápido, pois a CPU realiza menos instruções para cumprir a mesma tarefa. Vale notar que embora a taxa de “cache misses” suba, o tempo total ainda cai, pois a redução no número de instruções e o uso eficiente do pipeline compensam essas falhas.

Nos cenários de “ordem de linha”, “ordem de coluna” ou “blocagem”, sem otimização (-O0), blocagem tende a se sair um pouco melhor graças à localidade de dados. Mas ao usar -O3 ou -Ofast, todas as versões chegam perto de um mesmo patamar de desempenho.