GABRIEL DE OLIVEIRA PONTAROLO
GRR20203895
RODRIGO SAVIAM SOFFNER
GRR20205092

ANÁLISE DE DESEMPENHO DO MÉTODO DE NEWTON

Introdução a computação Científica - Trabalho 2

Relatório apresentado à disciplina Introdução a Computação Científica, do curso de Bacharelado em Ciência da Computação da Universidade Federal do Paraná – UFPR – como requisito parcial para aprovação na disciplina, sob a orientação do prof. Dr. Guilherme Alex Derenievicz e do prof. Me. Armando Luiz Nicolini Delgado.

CURITIBA
2022

SUMÁRIO

1. INTRODUÇÃO

O relatório a seguir consiste dos resultados obtidos durante as análises de desempenho de duas implementações do método de Newton, um algoritmo para encontrar os pontos críticos de funções de dimensão N, sendo que, em uma delas, foram utilizadas técnicas de otimização de código (especificadas na seção 4) com objetivo de obter um melhor desempenho do programa.

Para tal, foram feitas duas versões do método: Método de Newton Padrão, utilizando a técnica da Eliminação de Gauss com pivoteamento parcial para resolução do sistema linear gerado pela função, e o Método de Newton Inexato, utilizando a técnica iterativa de Gauss-Seidel para resolução do sistema. Com isso em mente, a aplicação escolhida para ser avaliada pelos métodos foi a Função de Rosenbrock, com suas dimensões variando entre 10 e 4096.

Ademais, a ferramenta *LIKWID* foi utilizada para realizar as medições de desempenho, com o foco em quatro métricas: *RTDSC Runtime*, para o tempo de execução; *DP* e *AVX DP* em MFLOPS/s, para operações aritméticas em ponto flutuante realizadas; *L2 miss ratio*, para a taxa de erro da cache e *L3 bandwidth* em MFLOPS/s*,* para a largura de banda de memória. Essas quatro métricas foram medidas em quatro trechos de código em ambas as duas implementações para cada um dos métodos. Tais trechos incluem: durante a aplicação completa do algoritmo; durante o cálculo dos vetores gradientes; durante o cálculo das matrizes Hessianas e durante a resolução dos sistemas lineares.

Desse modo, será possível verificar a efetividade que as otimizações de código tiveram sobre o desempenho do programa.

2. DESCRIÇÃO DO COMPUTADOR UTILIZADO

Saída do comando *likwid-topology -g -c*, descrevendo detalhadamente a máquina utilizada para compilar e executar todos os experimentos:

```
--------------------------------------------------------------------------------
CPU name:    Intel(R) Core(TM) i5-9400F CPU @ 2.90GHz
CPU type:    Intel Coffeelake processor
CPU stepping:    10
********************************************************************************
Hardware Thread Topology
********************************************************************************
Sockets:    1
Cores per socket:    6
Threads per core:    1
--------------------------------------------------------------------------------
HWThread    Thread    Core    Die    Socket    Available
0           0         0       0      0         *
1           0         1       0      0         *
2           0         2       0      0         *
3           0         3       0      0         *
4           0         4       0      0         *
5           0         5       0      0         *
--------------------------------------------------------------------------------
Socket 0:   ( 0 1 2 3 4 5 )
--------------------------------------------------------------------------------
********************************************************************************
Cache Topology
********************************************************************************
Level:   1
Size:   32 kB
Type:   Data cache
Associativity:   8
Number of sets:   64
Cache line size:    64
Cache type:   Non Inclusive
Shared by threads:    1
Cache groups:  ( 0 ) ( 1 ) ( 2 ) ( 3 ) ( 4 ) ( 5 )
--------------------------------------------------------------------------------
Level:   2
Size:   256 kB
Type:   Unified cache
Associativity:   4
```

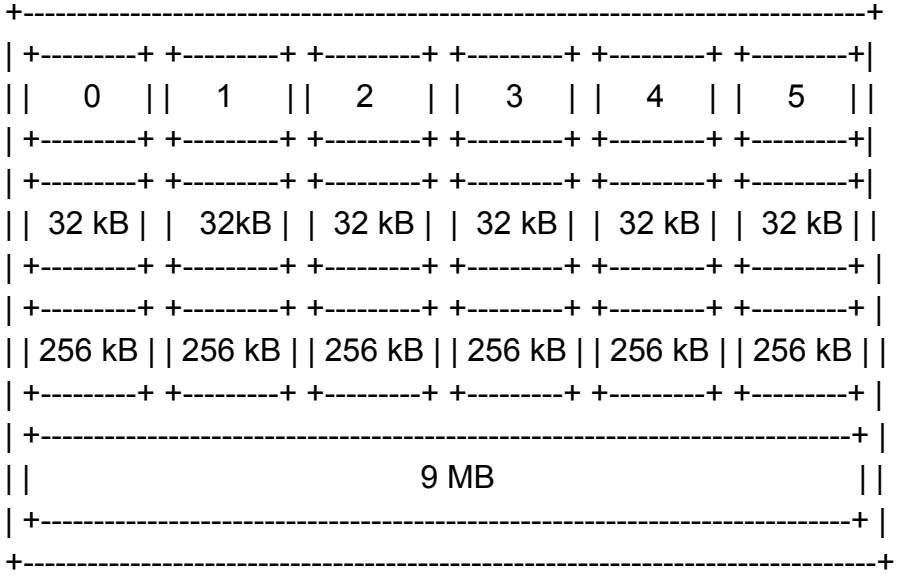
Number of sets: 1024
Cache line size: 64
Cache type: Non Inclusive
Shared by threads: 1
Cache groups: (0) (1) (2) (3) (4) (5)

Level: 3
Size: 9 MB
Type: Unified cache
Associativity: 12
Number of sets: 12288
Cache line size: 64
Cache type: Inclusive
Shared by threads: 6
Cache groups: (0 1 2 3 4 5)

****
NUMA Topology
****
NUMA domains: 1

Domain: 0
Processors: (0 1 2 3 4 5)
Distances: 10
Total memory: 15937.7 MB

****
Graphical Topology
****
Socket 0:



3. SOBRE A EXECUÇÃO DO PROGRAMA

Em posse dos códigos fontes, *scripts* e os outros arquivos que acompanham esse relatório no diretório *gop20-rss20/*, note que o código fonte da versão do programa sem as otimizações se encontra na pasta *pre-otimizado/*, dentro desse mesmo diretório, e a versão otimizada em *otimizado/*, ambos com seu próprio *Makefile* com as *flags* de compilação e limpeza (*make clean*). Além disso, os gráficos exibidos na seção 5 foram gerados pelo *script* em *python plot_graph.py,* a partir da saída do *LIKWID,* encontrado no diretório. Embora seja possível compilar e executá-los manualmente, o *script* em *bash run.sh* facilita o processo compilando, executando com os respectivos parâmetros do *LIKWID* para as quatro métricas, filtrando a saída e chamando o *script* para criação dos gráficos.

Para executá-lo, com a permissão de execução ativada, basta utilizar, dentro do diretório *gop20-rss20/*, o comando *./run.sh*. Os arquivos de saída do *LIKWID* serão direcionados para um diretório chamado *likwid_output/*, os arquivos contendo a saída em si do programa com os valores de f(x) alcançados por cada método em cada iteração serão criados em *newtonpc_output/* e, por fim, os gráficos serão gerados no diretório *graficos/*. Não é necessário criar previamente nenhum desses diretórios, pois o *script* já faz isso. A entrada para os programas é, por padrão, o arquivo *rosenbrock.txt* que abrange as Funções de Rosenbrock com os tamanhos entre 10 e 4096. Caso necessário, é possível especificar o arquivo de entrada utilizando *./run.sh {caminho para o arquivo de entrada}*.

4. OTIMIZAÇÕES EFETUADAS

As otimizações de código realizadas no programa incluem:

- A alocação dinâmica de matrizes foi feita de forma a posicioná-la na memória na forma de um único vetor contínuo, reduzindo o tempo de acesso e taxa de erro de cache. Otimização feita em todas as funções que utilizam da alocação de matrizes.
- Adição de um *padding* no tamanho dos vetores/matrizes, caso estes sejam uma potência de dois, o que evita o *cache trashing* reduzindo a taxa de erro de cache. Feito em todas as funções que utilizam alocação dinâmica de matriz/vetor.
- Vetorização de *loops*, permitindo que o compilador faça uso de instruções *AVX*, as quais fazem a soma de múltiplos espaços de memória ao mesmo tempo, reduzindo o tempo de execução. Utilizada na função privada *void _retrossubs(LINEAR_SYST *restrict syst)* do arquivo *gaussianElimination.c*, na função *void gaussSeidel(LINEAR_SYST *restrict syst);* da biblioteca *gaussSeidel.h* e no trecho onde é calculado o valor de X para próxima iteração dos Métodos de Newton Padrão e Inexato, na função *void NewtonPadrao(FUNCTION *restrict func);* da biblioteca *newtonPadrao.h* e na função *void NewtonInexato(FUNCTION *restrict func);* da biblioteca *newtonInexato.h*, respectivamente.
- Uso da palavra chave *restrict*, indicando para o compilador que não há dois ponteiros diferentes apontando para a mesma localização específica de memória, permitindo diversas outras otimizações (como o uso de *AVX*). Utilizada em todas as funções que tem como argumento um ponteiro.
- Remoção de operações complexas onde era possível:
    - Substituição de todas as chamadas da função *pow(x, y)* da biblioteca *math.h* por uma multiplicação, pois a primeira é uma operação custosa em termos de ciclos de *clock.*
    - Função *double norma(double *array, int size);* da biblioteca *utils.h*, utilizada frequentemente no programa para o cálculo da norma euclidiana do vetor e que utilizava a função *sqrt(x)* de *math.h*, foi substituída por *double sq_norma(double *array, int size);* a qual retorna

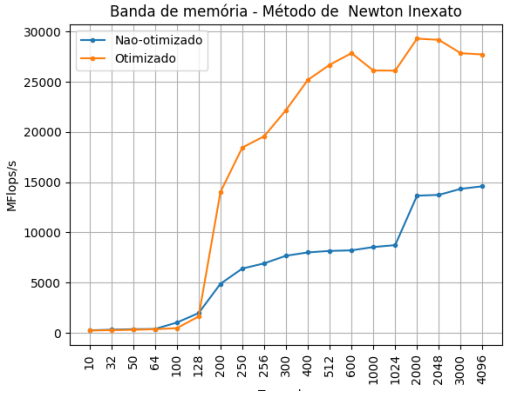
o quadrado da norma euclidiana e não faz uso da função, pois essa também é custosa em termos de ciclos. Foram feitas as respectivas adaptações nos trechos que código que utilizavam essa função.

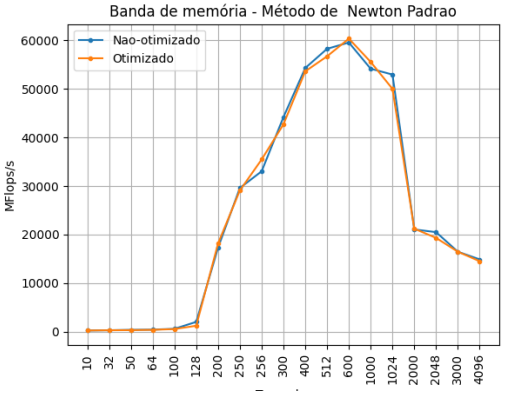
- Remoção de condicionais dentro do *loop*, pois eles prejudicam o *pipelining*. Foi utilizado duas vezes na função *void gaussSeidel(LINEAR_SYST *restrict syst);* da biblioteca *gaussSeidel.h* (juntamente com a vetorização).

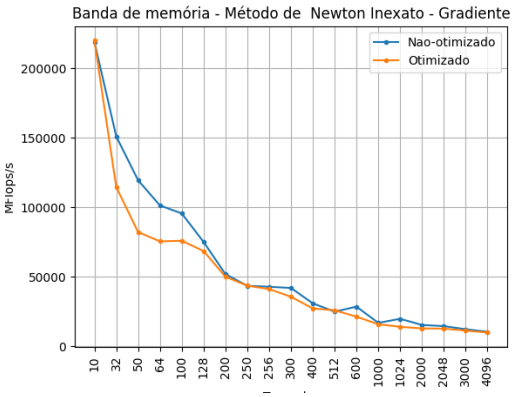
É importante ressaltar que, ambas as duas versões, otimizada e não otimizada, foram executadas utilizando as mesmas *flags* de compilação do gcc, especificamente *-std=c99 -Wall -O3 -mavx -march=native -fstrict-aliasing*, desse modo, as diferenças de performance observadas serão devido às mudanças no código unicamente. Também é importante observar que não houve nenhuma mudança nas funções utilizadas para calcular o vetor gradiente e a matriz hessiana para a Função de Rosenbrock, logo as diferenças de performance serão devido a forma como elas foram chamadas e as disposições das estruturas de dados na memória.

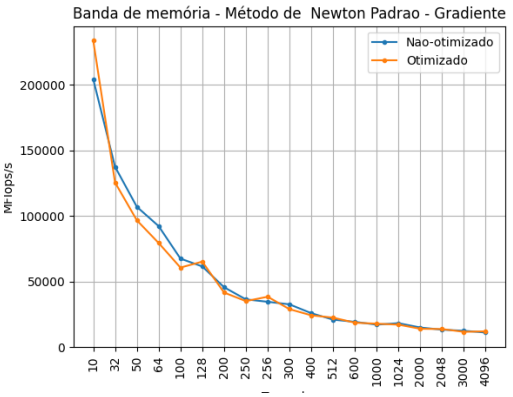
5. GRÁFICOS DE COMPARAÇÃO DE PERFORMANCE

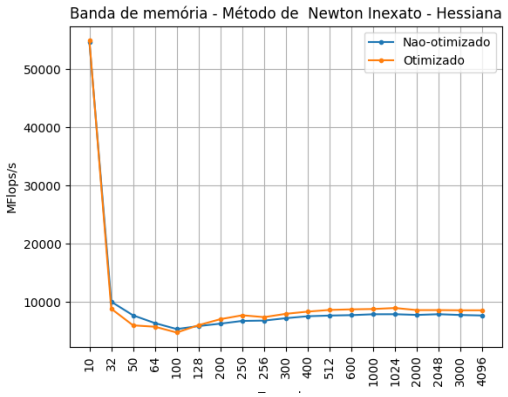
A seguir temos o resultado dos testes efetuados. Todos os gráficos possuem a dimensão da função no eixo das abscissas e a respectiva métrica no eixo das ordenadas, assim como os valores do mesmo trecho executado nas duas versões do programa.

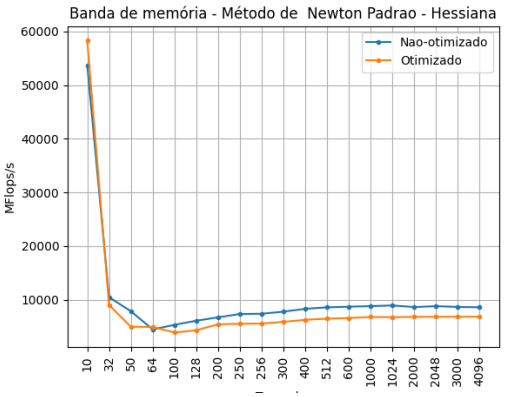

Banda de memória - Método de Newton Inexato
Nao-otimizado
Otimizado
MFlops/s

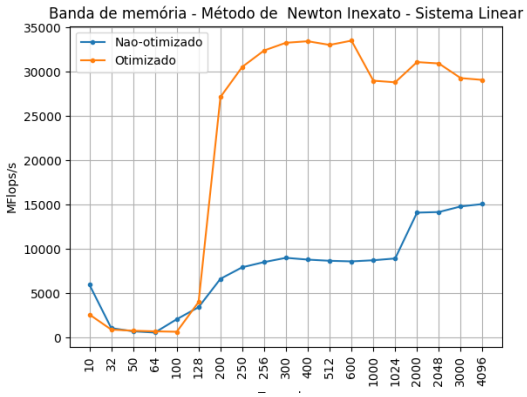

Banda de memória - Método de Newton Padrao
Nao-otimizado
Otimizado
MFlops/s

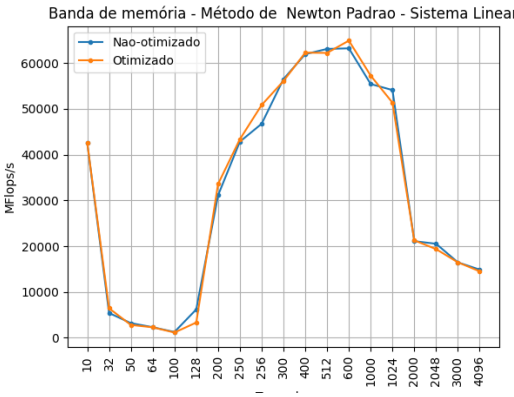

Banda de memória - Método de Newton Inexato - Gradiente
Nao-otimizado
Otimizado
MFlops/s

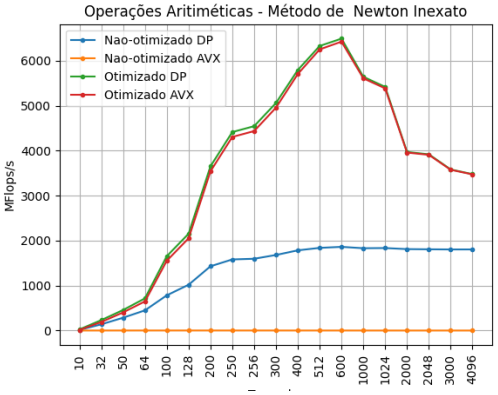

Banda de memória - Método de Newton Padrao - Gradiente
Nao-otimizado
Otimizado
MFlops/s

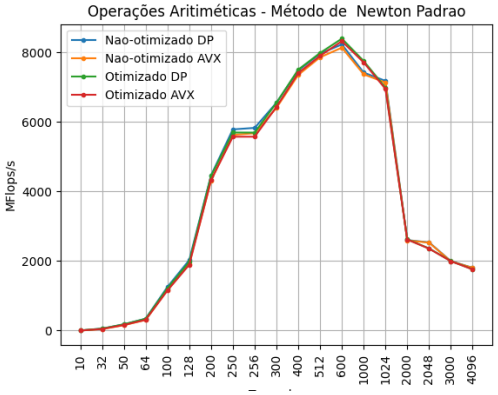

Banda de memória - Método de Newton Inexato - Hessiana
Nao-otimizado
Otimizado
MFlops/s

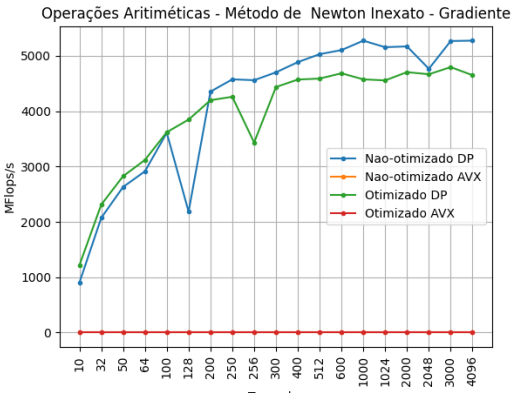

Banda de memória - Método de Newton Padrao - Hessiana
Nao-otimizado
Otimizado
MFlops/s

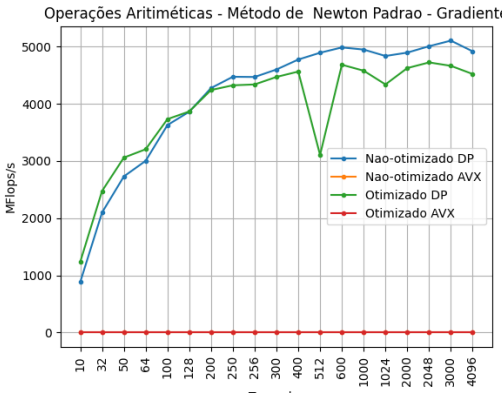

Banda de memória - Método de Newton Inexato - Sistema Linear
Nao-otimizado
Otimizado
MFlops/s

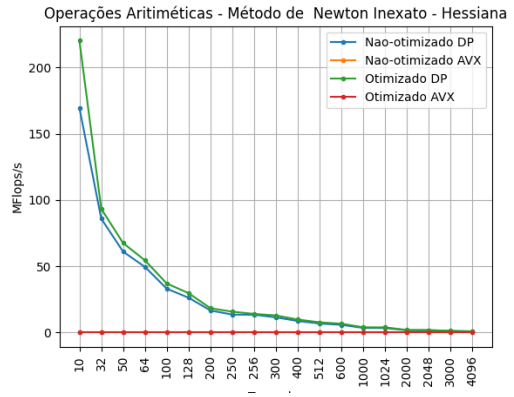

Banda de memória - Método de Newton Padrao - Sistema Linear
Nao-otimizado
Otimizado
MFlops/s

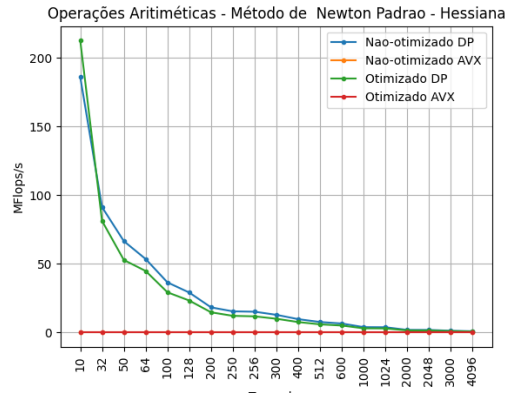

Operações Aritiméticas - Método de Newton Inexato
Nao-otimizado DP
Nao-otimizado AVX
Otimizado DP
Otimizado AVX
MFlops/s

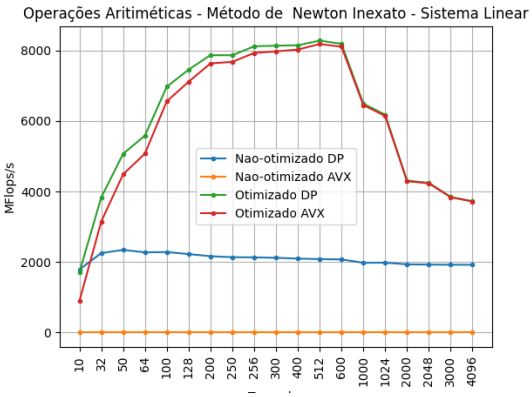

Operações Aritiméticas - Método de Newton Padrao
Nao-otimizado DP
Nao-otimizado AVX
Otimizado DP
Otimizado AVX
MFlops/s

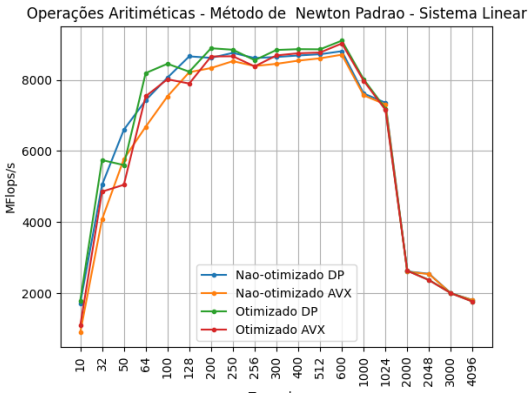

Operações Aritiméticas - Método de Newton Inexato - Gradiente
Nao-otimizado DP
Nao-otimizado AVX
Otimizado DP
Otimizado AVX
MFlops/s

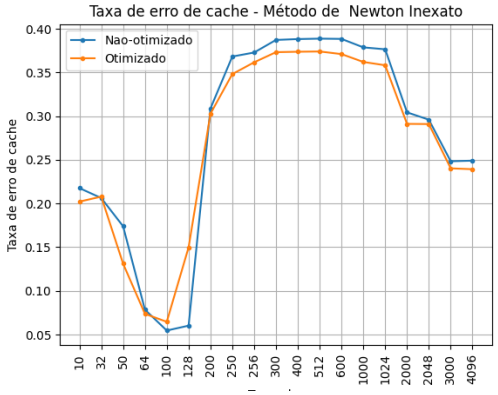

Operações Aritiméticas - Método de Newton Padrao - Gradiente
Nao-otimizado DP
Nao-otimizado AVX
Otimizado DP
Otimizado AVX
MFlops/s

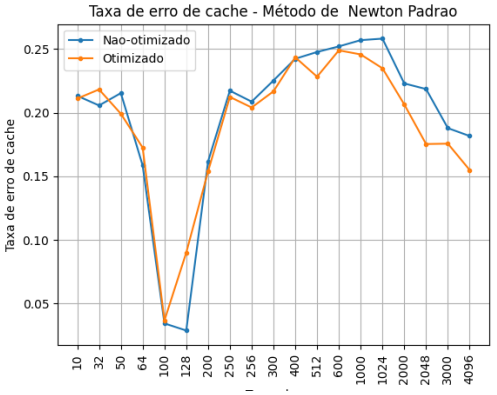

Operações Aritiméticas - Método de Newton Inexato - Hessiana
Nao-otimizado DP
Nao-otimizado AVX
Otimizado DP
Otimizado AVX
MFlops/s

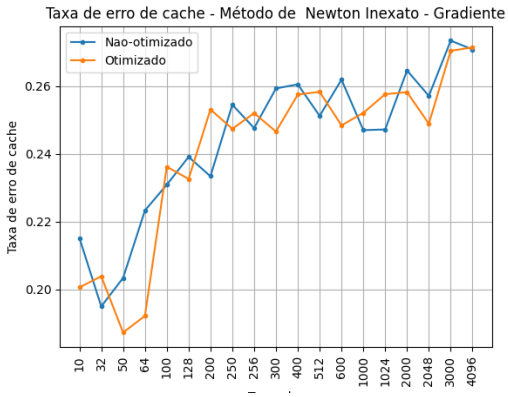

Operações Aritiméticas - Método de Newton Padrao - Hessiana
Nao-otimizado DP
Nao-otimizado AVX
Otimizado DP
Otimizado AVX
MFlops/s

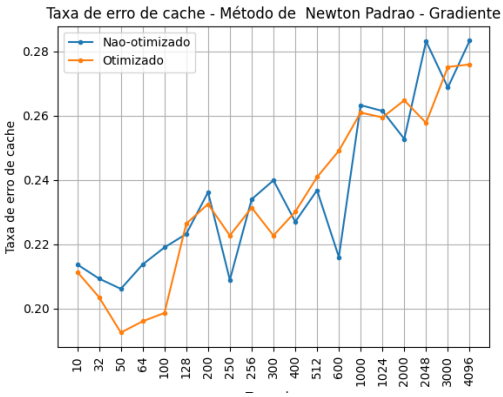

Operações Aritiméticas - Método de Newton Inexato - Sistema Linear
Nao-otimizado DP
Nao-otimizado AVX
Otimizado DP
Otimizado AVX
MFlops/s

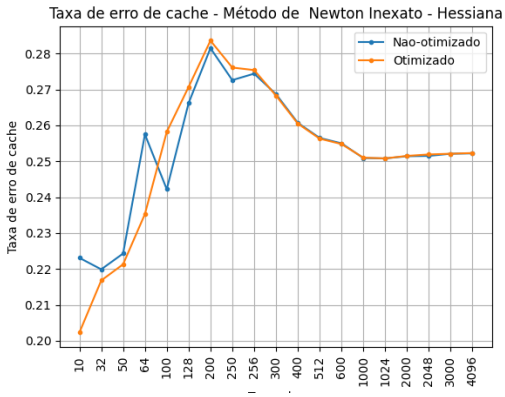

Operações Aritiméticas - Método de Newton Padrao - Sistema Linear
Nao-otimizado DP
Nao-otimizado AVX
Otimizado DP
Otimizado AVX
MFlops/s

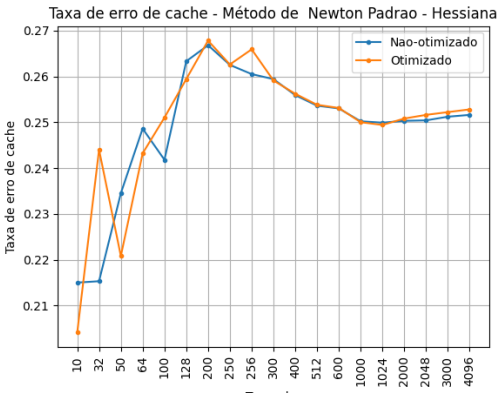

Taxa de erro de cache - Método de Newton Inexato
Nao-otimizado
Otimizado
Taxa de erro de cache

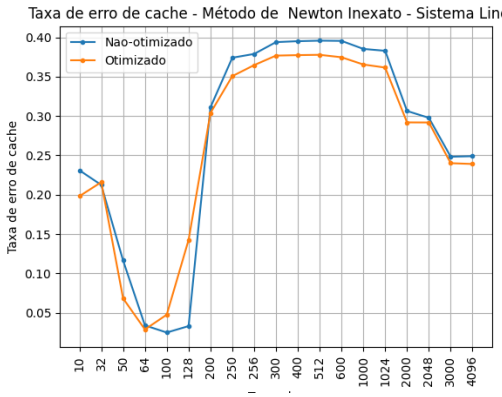

Taxa de erro de cache - Método de Newton Padrao
Nao-otimizado
Otimizado
Taxa de erro de cache

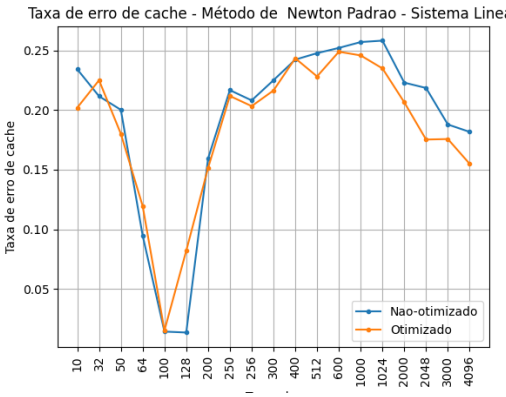

Taxa de erro de cache - Método de Newton Inexato - Gradiente
Nao-otimizado
Otimizado
Taxa de erro de cache

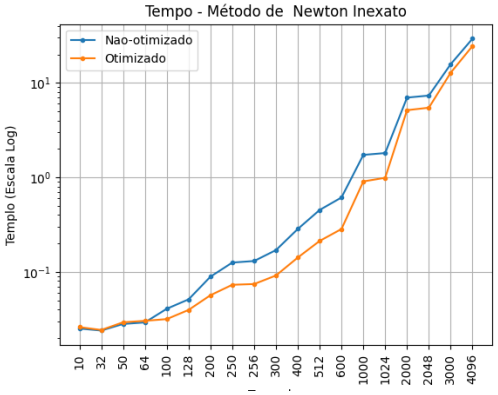

Taxa de erro de cache - Método de Newton Padrao - Gradiente
Nao-otimizado
Otimizado
Taxa de erro de cache

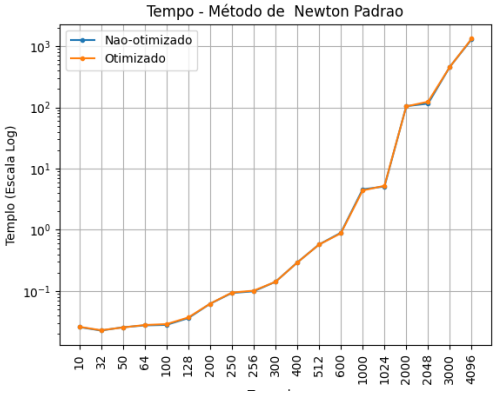

Taxa de erro de cache - Método de Newton Inexato - Hessiana
Nao-otimizado
Otimizado
Taxa de erro de cache

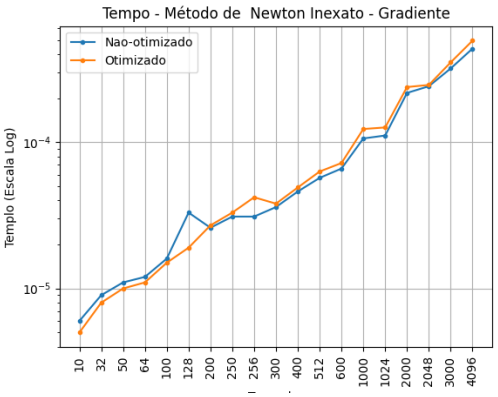

Taxa de erro de cache - Método de Newton Padrao - Hessiana
Nao-otimizado
Otimizado
Taxa de erro de cache

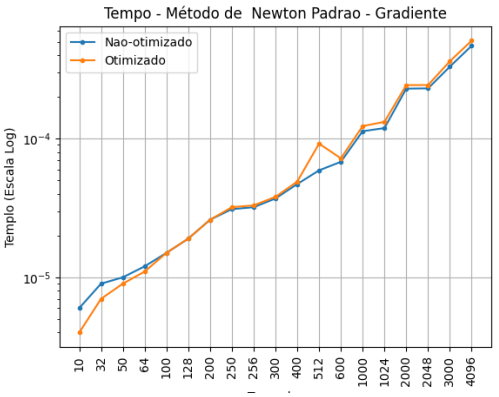

Taxa de erro de cache - Método de Newton Inexato - Sistema Linear
Nao-otimizado
Otimizado
Taxa de erro de cache

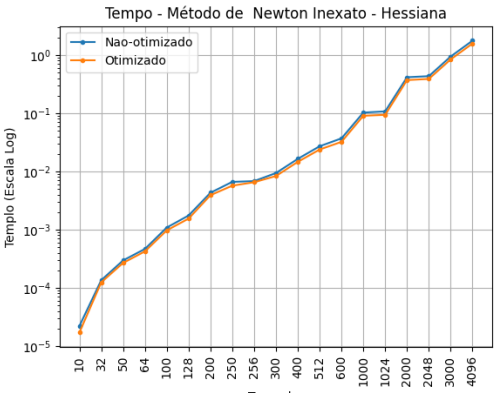

Taxa de erro de cache - Método de Newton Padrao - Sistema Linear
Nao-otimizado
Otimizado
Taxa de erro de cache

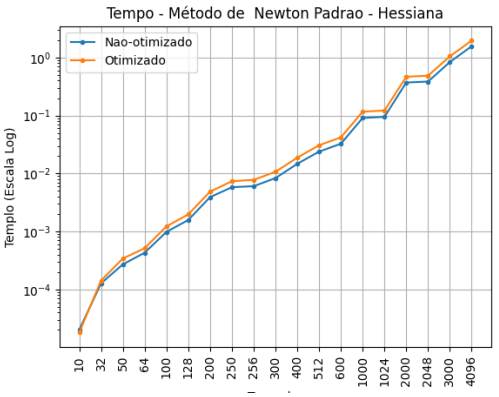

Tempo - Método de Newton Inexato
Nao-otimizado
Otimizado
Templo (Escala Log)

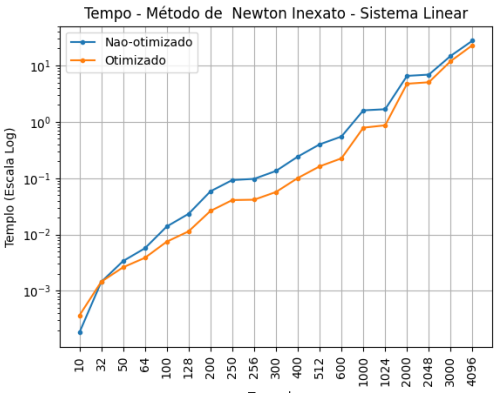

Tempo - Método de Newton Padrao
Nao-otimizado
Otimizado
Templo (Escala Log)

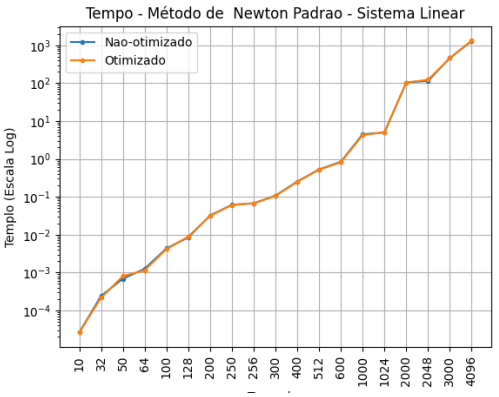

Tempo - Método de Newton Inexato - Gradiente
Nao-otimizado
Otimizado
Templo (Escala Log)


Tempo - Método de Newton Padrao - Gradiente
Nao-otimizado
Otimizado
Templo (Escala Log)


Tempo - Método de Newton Inexato - Hessiana
Nao-otimizado
Otimizado
Templo (Escala Log)


Tempo - Método de Newton Padrao - Hessiana
Nao-otimizado
Otimizado
Templo (Escala Log)


Tempo - Método de Newton Inexato - Sistema Linear
Nao-otimizado
Otimizado
Templo (Escala Log)


Tempo - Método de Newton Padrao - Sistema Linear
Nao-otimizado
Otimizado
Templo (Escala Log)

É possível observar que a adição de operações *AVX* para o cálculo do Gauss-Seidel provocou uma diferença grande na performance, como é mostrado pelos gráficos de tempo, banda de memória e operações aritméticas do sistema linear no Método de Newton Inexato. Em contrapartida, a adição do mesmo na retrossubstituição não se mostrou visivelmente eficaz, como é visto nos gráficos.

Como foi mencionado antes, não houve uma otimização direta no código das funções que calculam o gradiente e a hessiana. De fato, as curvas do gŕafico se mostram consideravelmente similares no quesito de tempo, operações aritméticas e largura de banda. Entretanto, é possível observar alterações em alguns picos e vales nos tamanhos que são potências de 2, algo que fica bem visível nos gráficos da taxa de erro de cache. Isso mostra o efeito da adição no *padding* na alocação de memória para redução de *cache trashing*.