

UNIVERSIDADE FEDERAL DO PARANÁ

RELATÓRIO DO TRABALHO 2  
ANÁLISE DE DESEMPENHO

Curitiba - PR

2015

UNIVERSIDADE FEDERAL DO PARANÁ

HENRIQUE COLODETTI ESCANFERLA  
ISRAEL BARTH RUBIO

RELATÓRIO DO TRABALHO 2  
ANÁLISE DE DESEMPENHO

Trabalho apresentado á disciplina de  
Introdução a Computação Científica, do curso  
de Bacharelado de Ciência da Computação,  
sob orientação do professor Daniel  
Weingaertner.

Curitiba - PR

2015

# SUMÁRIO

1. ANÁLISE GERAL .....	4
1.1. Arquitetura da Máquina utilizada .....	4
1.2. Limite de Discretização do Espaço Amostral .....	7
1.3. Gráfico Tempo de Execução por Discretização .....	7
2. Análise das funções Gauss Seidel e cálculo do Resíduo .....	8
2.1. Função Gauss Seidel .....	8
2.1.1. Número de FLOPS por tamanho da discretização .....	8
2.1.2. Memória utilizada por tamanho da discretização .....	8
2.1.3. Gráficos do likwid sem e com otimização, -O0 e -O3 e análise do Gauss Seidel .....	9
2.2. Função do Cálculo do Resíduo .....	11
2.2.1. Nº de FLOPS no Cálculo do Resíduo .....	11
2.2.2. Memória utilizada no Cálculo do Resíduo .....	12
2.2.3. Gráficos do likwid sem e com otimização, -O0 e -O3 e análise do Gauss Seidel .....	12
2. Considerações Finais .....	14
3. Referências .....	15

## 1. ANÁLISE GERAL

### 1.1. Arquitetura da Máquina utilizada

Segue abaixo a arquitetura da máquina utilizada reportado pelo likwid:

-----

CPU type: Intel Core Westmere processor

\*\*\*\*\*

Hardware Thread Topology

\*\*\*\*\*

Sockets: 2

Cores per socket: 6

Threads per core: 2

-----

HWThread	Thread	Core	Socket
0	0	0	0
1	0	1	0
2	0	2	0
3	0	8	0
4	0	9	0
5	0	10	0
6	0	0	1
7	0	1	1
8	0	2	1
9	0	8	1
10	0	9	1

11	0	10	1
12	1	0	0
13	1	1	0
14	1	2	0
15	1	8	0
16	1	9	0
17	1	10	0
18	1	0	1
19	1	1	1
20	1	2	1
21	1	8	1
22	1	9	1
23	1	10	1

-----

Socket 0: ( 0 12 1 13 2 14 3 15 4 16 5 17 )

Socket 1: ( 6 18 7 19 8 20 9 21 10 22 11 23 )

-----

\*\*\*\*\*

## Cache Topology

\*\*\*\*\*

Level: 1

Size: 32 kB

Cache groups: ( 0 12 ) ( 1 13 ) ( 2 14 ) ( 3 15 ) ( 4 16 ) ( 5 17 ) ( 6 18 ) ( 7 19 ) ( 8 20 ) ( 9 21 ) ( 10 22 ) ( 11 23 )

-----  
Level: 2

Size: 256 kB

Cache groups: ( 0 12 ) ( 1 13 ) ( 2 14 ) ( 3 15 ) ( 4 16 ) ( 5 17 ) ( 6 18 ) ( 7 19 ) ( 8 20 ) ( 9 21 ) ( 10 22 ) ( 11 23 )  
-----

Level: 3

Size: 12 MB

Cache groups: ( 0 12 1 13 2 14 3 15 4 16 5 17 ) ( 6 18 7 19 8 20 9 21 10 22 11 23 )  
-----

\*\*\*\*\*

## NUMA Topology

\*\*\*\*\*

NUMA domains: 2

-----  
Domain 0:

Processors: 0 1 2 3 4 5 12 13 14 15 16 17

Relative distance to nodes: 10 21

Memory: 1388.66 MB free of total 24105.4 MB  
-----

Domain 1:

Processors: 6 7 8 9 10 11 18 19 20 21 22 23

Relative distance to nodes: 21 10

Memory: 13355.8 MB free of total 24190 MB

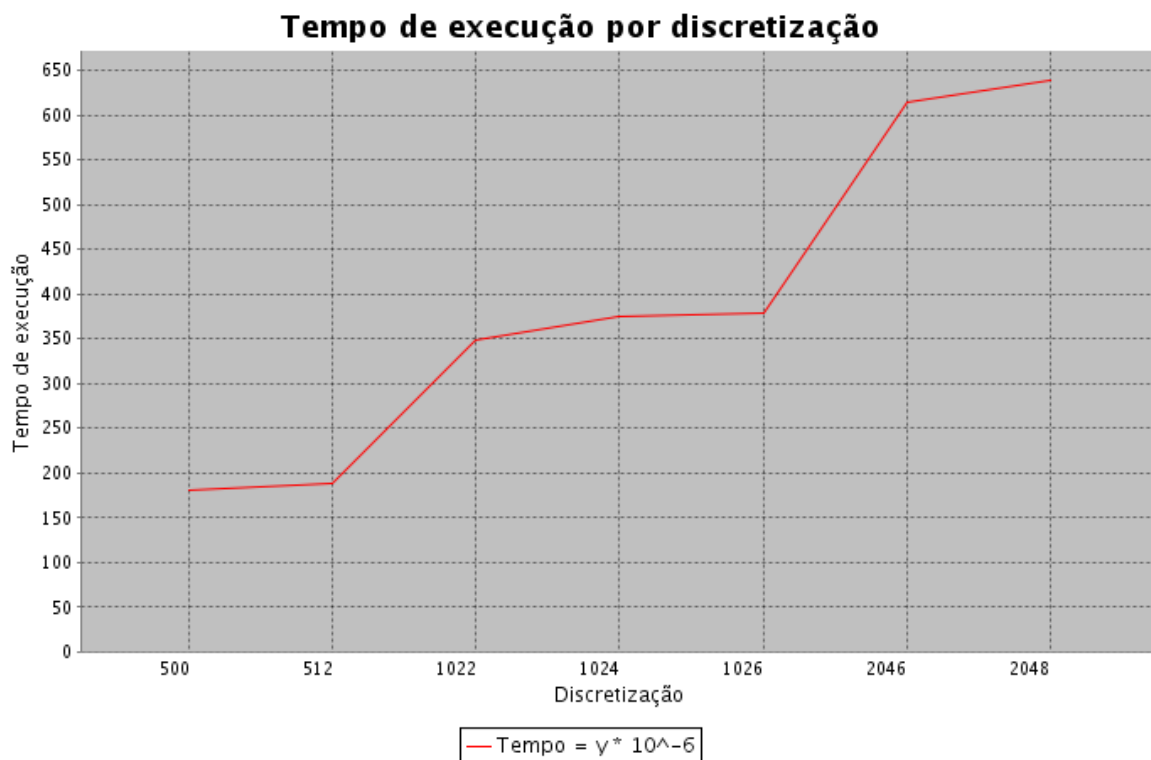
## 1.2. Limite de Discretização do Espaço Amostral

Desprezando as partes constantes no custo de memória utilizada no algoritmo, temos que o número de pontos da discretização, dado por  $n_x$  vezes  $n_y$ , vezes 8 (memória de um ponto) resulta na quantidade teórica de memória necessária para executar o programa. A máquina em questão possui  $24190 * 1024 = 24770560$  bytes.  $24770560/8 = 3096320$  pontos é o máximo que a máquina é capaz de computar.

Na prática, temos de considerar todo o custo constante, em relação ao tamanho da discretização, de memória e a memória livre atual da máquina que varia de acordo com seu uso no momento.

## 1.3. Gráfico Tempo de Execução por Discretização

Segue abaixo o gráfico do tempo de execução por tamanho dos pontos da discretização. A grandeza dos tempos mostrados é de  $10^{-6}$  segundos.



## 2. Análise das funções Gauss Seidel e cálculo do Resíduo

### 2.1. Função Gauss Seidel

#### 2.1.1. Número de FLOPS por tamanho da discretização

Desconsiderando os FLOPS constantes independente da forma como o espaço amostral é discretizado. Seja o  $N^o$  de pontos na horizontal igual ao  $N^o$  de pontos na vertical que chamaremos de  $X$ .

Temos  $X - 1$  FLOPS resultante do  $N^o$  de vezes que a função pula o ghost layer utilizado em torno dos pontos discretizados. Isto ocorre dentro do loop da seção de código da iteração de Gauss Seidel mas ocorre  $X - 1$  vezes.

O loop realiza  $X^2 - 4 * X + 4$  iterações. Veja que temos  $X^2$  pontos a serem iterados e 4 bordas do ghost layer de tamanho  $X$ . Quando subtraímos  $4 * X$  de  $X^2$ , esquecemos que estamos contando os 4 pontos dos 4 cantos do ghost layer 2 vezes, então, somamos 4 para corrigir a conta.

Dentro do loop em questão, fazemos sempre  $1 + 10 + 46$  ( $N^o$  de FLOPS aproximado da função seno) + 5 ( $N^o$  de FLOPS aproximado da função seno hiperbólico) + 4. O 1º FLOP é do comando " $dx += \text{delta\_x}$ ", os  $10 + 46 + 5$  são os FLOPS do cálculo de um ponto da discretização e os 4 ultimos são FLOPS necessários para comparar o novo valor do ponto com o anterior e interagir com o fator  $W$  do método SOR para obtermos o próximo valor do ponto.

Concluindo, temos  $X - 1 + 66 * (X^2 - 4 * X + 4)$  FLOPS para o Gauss Seidel.

#### 2.1.2. Memória utilizada por tamanho da discretização

Variáveis inteiras de 4 bytes:  $l, k, ni, j, i, nx, ci, 1, ci2, ci3 \implies 4 * 9 = 36$  bytes.

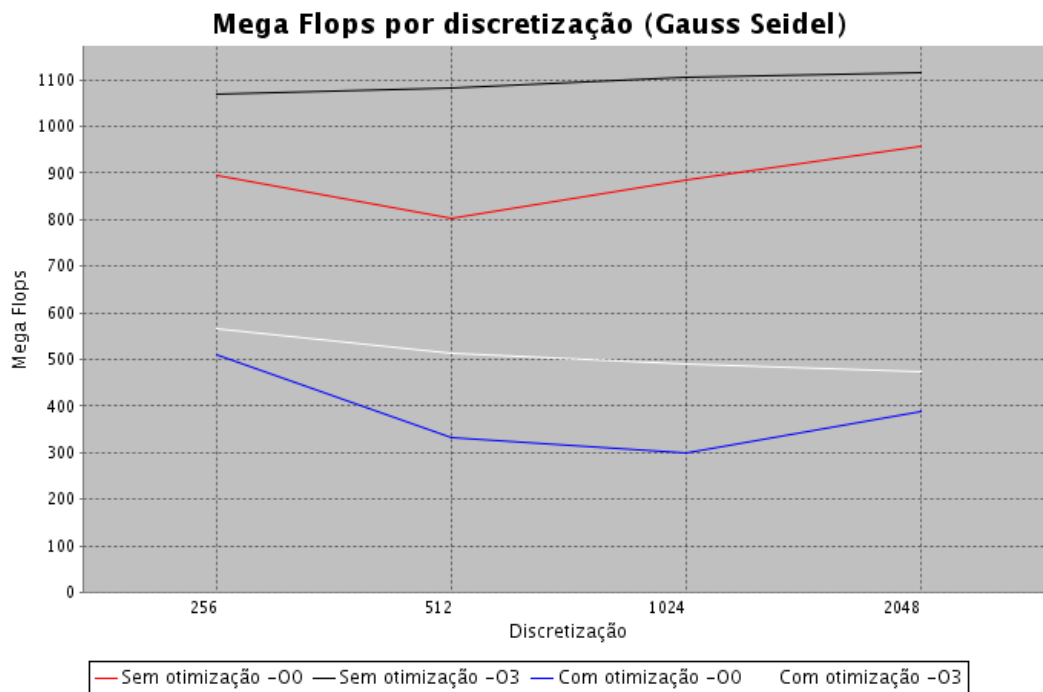
Váriaveis double de 8 bytes:  $dx, \text{delta\_x}, dy, \text{delta\_y}, \text{pt\_ant}, \text{inc}[X^2]$  (considerando  $n_x = n_y = X$ ),  $\text{inv\_delta\_x2}, \text{inv\_delta\_y}, k2, \_2PI, cf1, w \implies 11 * 8 + X^2 * 8$  bytes.



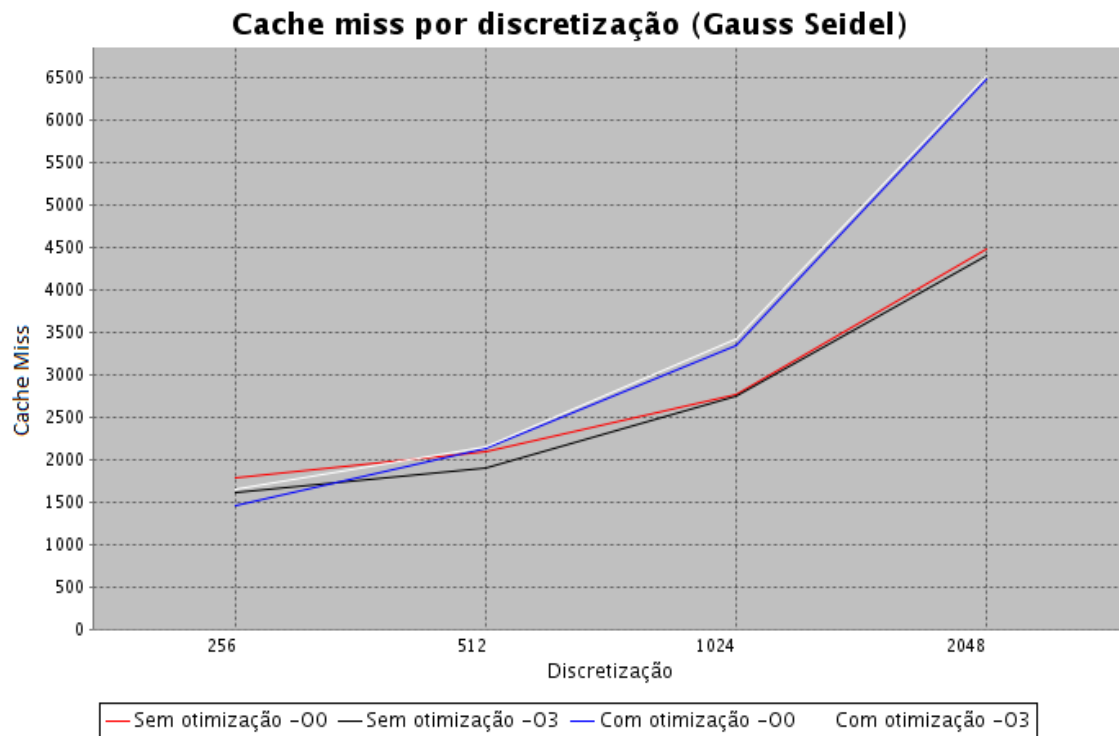
Concluindo, temos  $36 + 11 * 8 + X^2 * 8 = 124 + X^2 * 8$  bytes para o Gauss Seidel.

### 2.1.3. Gráficos do likwid sem e com otimização, -O0 e -O3 e análise do Gauss Seidel

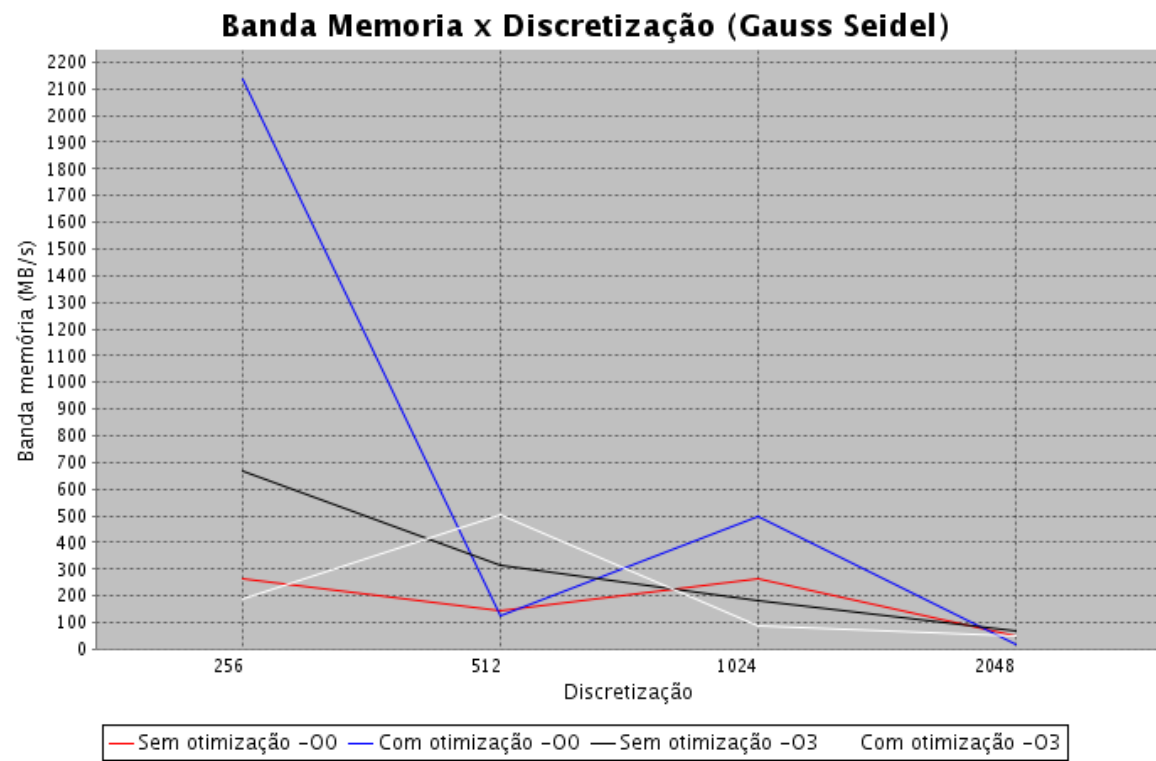
Segue abaixo o gráfico mostrando a taxa MEGA FLOPS por segundo:



Segue abaixo o gráfico mostrando o N° de cache miss:



Segue abaixo o gráfico mostrando a taxa de banda de memória:



Observando os gráficos, podemos perceber que, com o tamanho da discretização, as taxas de MEGA FLOPS por segundo e a banda de memória aumentam e diminuem. Estes valores não são bons para indicar se a mudança da performance, utilizando as opções de compilação e as técnicas de otimização, foi para melhor, pior ou se não ocorreu algo significativo. Tais taxas precisam do tempo médio de execução por discretização para obtermos mais precisamente o N° de FLOPS e a quantidade de memória tráfegada quando compilamos no modo -O0, -O3 com e sem otimização.

Veja que o N° de cache miss não apresenta mudanças significativas em relação ao modo de compilação mas vemos um aumento deles com a otimização. Isso deve a técnica que usamos para otimizar o código chamado de look up table. Ela nos ajuda a diminuir o N° de FLOPS com o custo de acessar mais memória que, eventualmente esta ou não na cache do processador.

A grande melhora da otimização esta no N° de FLOPS principalmente por serem feitas identicamente dentro de um loop que chama funções demasiadamente custosas (seno e seno hiperbólico), mas isto não é devidamente apresentado na taxa de MEGA FLOPS por segundo pois isto só demonstra o quanto estamos usando da potência de cálculo de FLOPS por segundo do processador da máquina utilizada nos testes. O aumento ou diminuição de tal taxa não necessariamente indica que a performance do código melhorou ou piorou.

Parece bom que a taxa tenha aumentado pois pode significar que o código tem maior rendimento em relação aos FLOPS no processador, mas simultaneamente pode indicar que o código esta usando mais FLOPS do que antes da modificação. De mesmo modo, pode parecer bom que o código esteja tráfegando mais memória, mas simultaneamente pode indicar que o código esta manipulando mais memória do que antes da modificação.

## **2.2. Função do Cálculo do Resíduo**

### **2.2.1. N° de FLOPS no Cálculo do Resíduo**

Temos  $X - 1$  FLOPS resultante do N° de vezes que a função pula o ghost layer utilizado em torno dos pontos discretizados. Isto ocorre dentro do loop da seção de código da iteração do cálculo do resíduo mas ocorre  $X - 1$  vezes.

O loop realiza  $X^2 - 4 * X + 4$  iterações. Veja que temos  $X^2$  pontos a serem iterados e 4 bordas do ghost layer de tamanho  $X$ . Quando subtraímos  $4 * X$  de  $X^2$ ,

esquecemos que estamos contando os 4 pontos dos 4 cantos do ghost layer 2 vezes, então, somamos 4 para corrigir a conta.

Dentro do loop em questão, fazemos sempre  $2 + 11 + 46$  (Nº de FLOPS aproximado da função seno) + 5 (Nº de FLOPS aproximado da função seno hiperbólico) + 2 + 10~30 (Nº de FLOPS aproximado da função sqrt que tira a raiz). O 1º FLOP é do comando "dx += delta\_x", o 2º FLOP é a operação "-" para trocar o sinal de parte da equação do problema para obter o resíduo, os  $11 + 46 + 5$  são os FLOPS do cálculo de um ponto da discretização e os 2 seguintes são FLOPS necessários para elevar ao quadrado e somar com o resíduo atual para podermos tirar a norma euclidiana do vetor dos resíduos. A função sqrt que tira a raiz nos traz os últimos 10~30 FLOPS realizados por iteração do loop do cálculo do resíduo.

Concluindo, temos  $X - 1 + 76 \sim 96 * (X^2 - 4 * X + 4)$  FLOPS para o cálculo do resíduo.

### 2.2.2. Memória utilizada no Cálculo do Resíduo

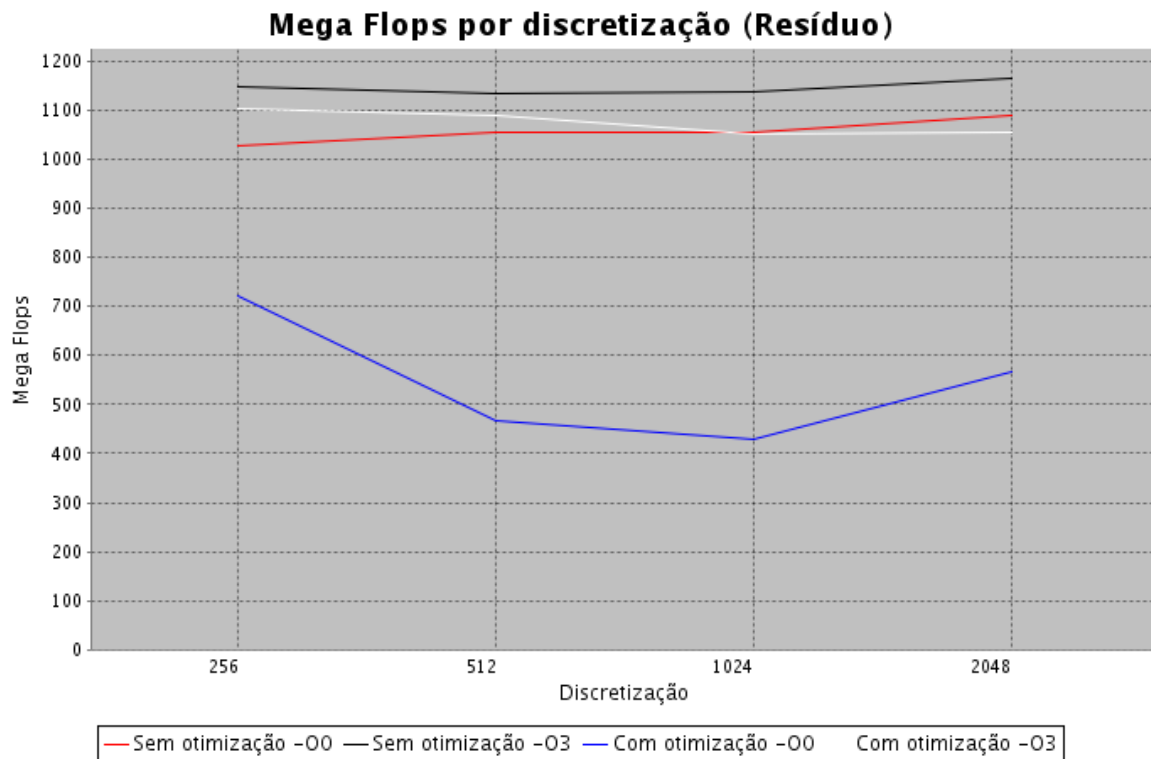
Variáveis inteiras: i, j, nx, ci1, ci2, ci3 =  $6 * 4 = 24$  bytes.

Variáveis double: dx, delta\_x, dy, delta\_y, aux\_res, k2, \_2PI, inv\_delta\_x2, inv\_delta\_y2, cf2, r + inc[X<sup>2</sup>] =  $11 * 8 + X^2 * 8$  bytes.

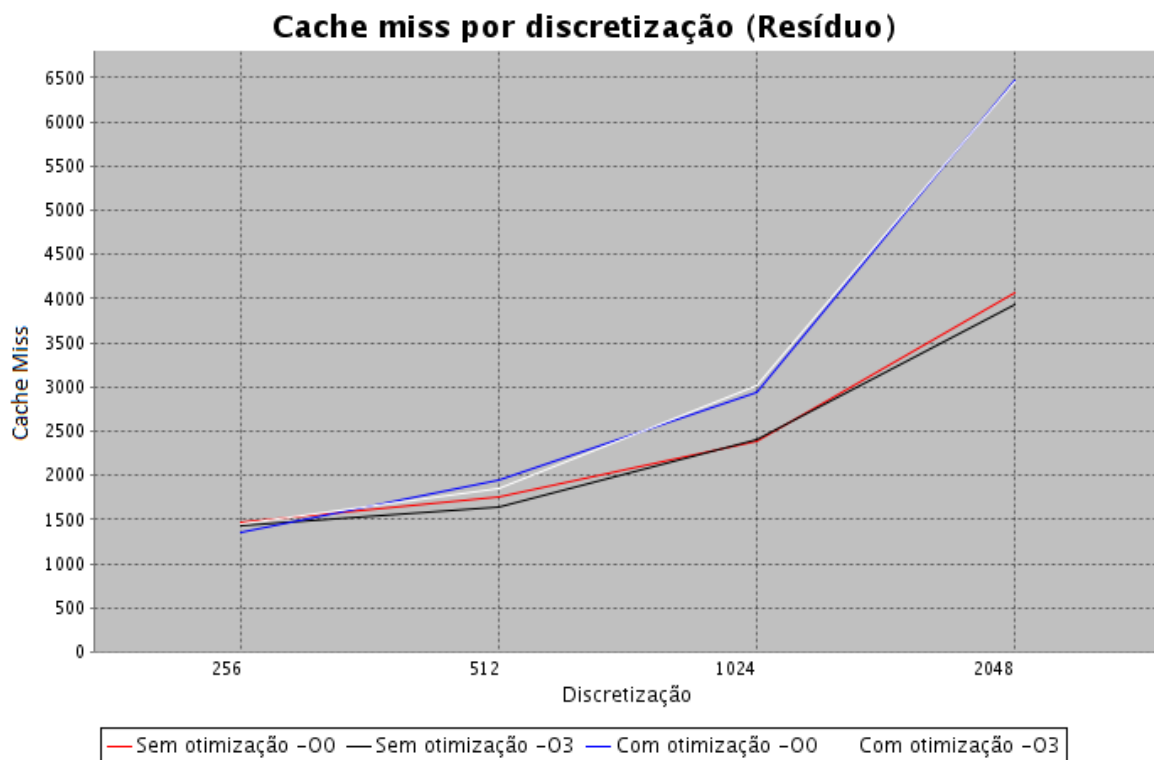
Concluindo, temos  $24 + 11 * 8 + X^2 * 8 = 112 + X^2 * 8$  bytes.

### 2.2.3. Gráficos do likwid sem e com otimização, -O0 e -O3 e análise do Gauss Seidel

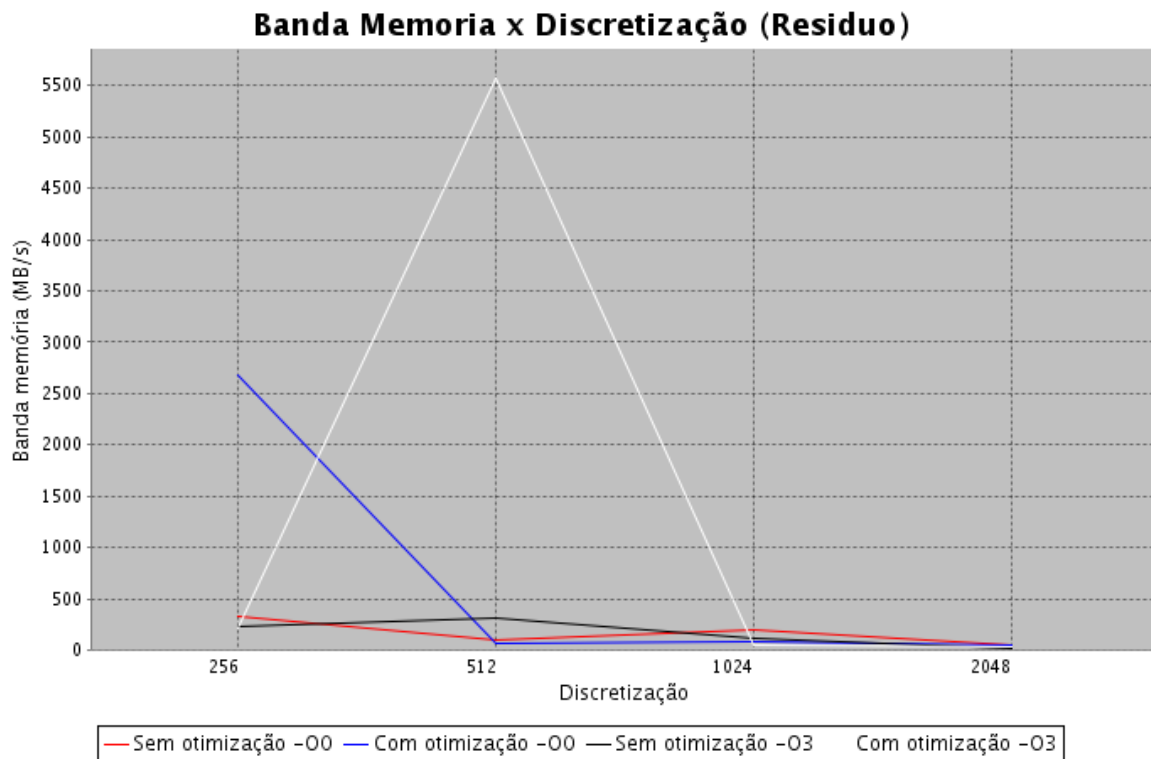
Segue abaixo o gráfico mostrando a taxa MEGA FLOPS por segundo:



Segue abaixo o gráfico mostrando o N° de cache miss:



Segue abaixo o gráfico mostrando a taxa de banda de memória:



A análise aqui é exatamente idêntica à análise dos gráficos da função que executa o método de Gauss Seidel na seção 2.1.3.

## 2. Considerações Finais

As ferramentas que o compilador oferece e as bibliotecas especiais existentes para a otimização da performance do código tem sua eficiência dependente de vários fatores envolvidos. Alguns deles são o processador utilizado nos testes, o algoritmo do código, a forma como ele é implementado e a forma da otimização experimentada. Devemos analisar a situação por completo para tomarmos as decisões corretas sobre o que deve ser otimizado e o que não traz mudanças significativas de performance.

O likwid é um componente muito importante para sabermos exatamente qual seção da máquina está engasgado com a execução do código e nos focarmos em tal seção para otimizá-lo ao máximo. Com o likwid, podemos saber o que é mais importante diminuir: FLOPS, Cache Miss, a memória trafegada ou outra métrica de performance. Isto aponta uma melhor direção de onde e o que deveria melhorar no código para aumentar a performance.

### 3. Referências

- 1) Documentação oficial do likwid: <https://github.com/rze-likwid/likwid>
- 2) Gerador de Gráficos: <http://www.barchart.be>