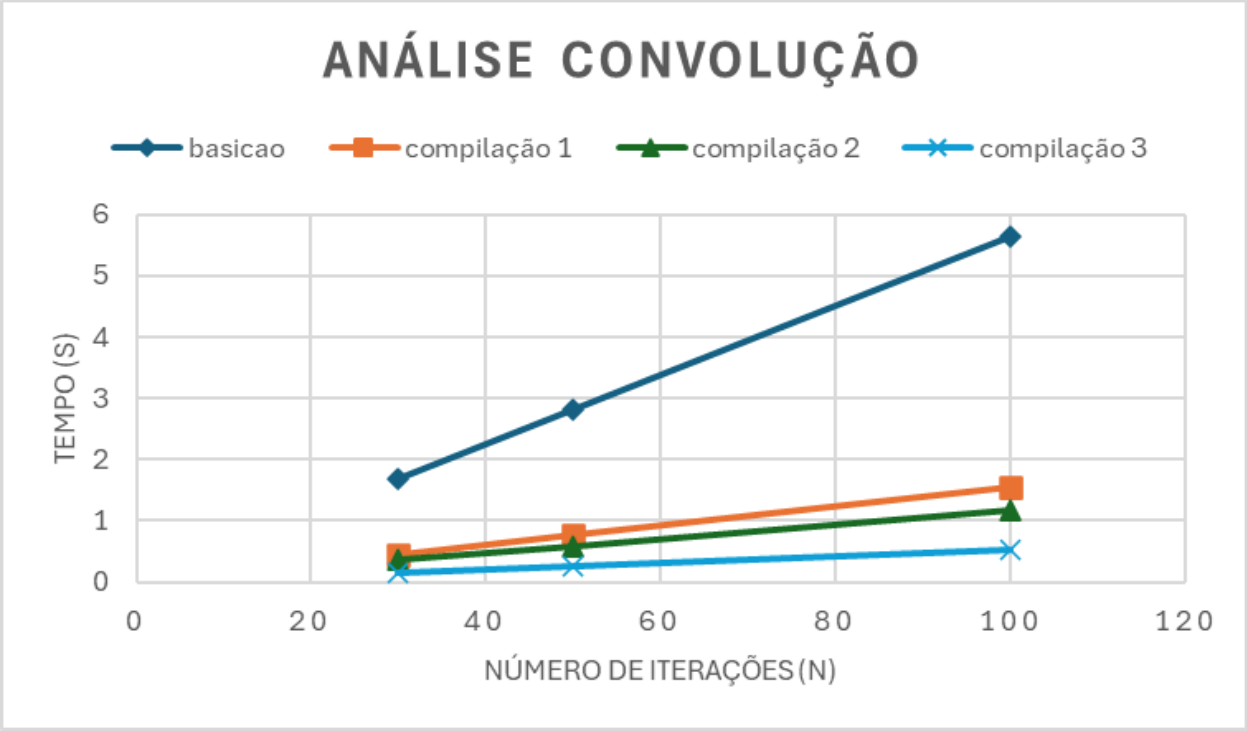
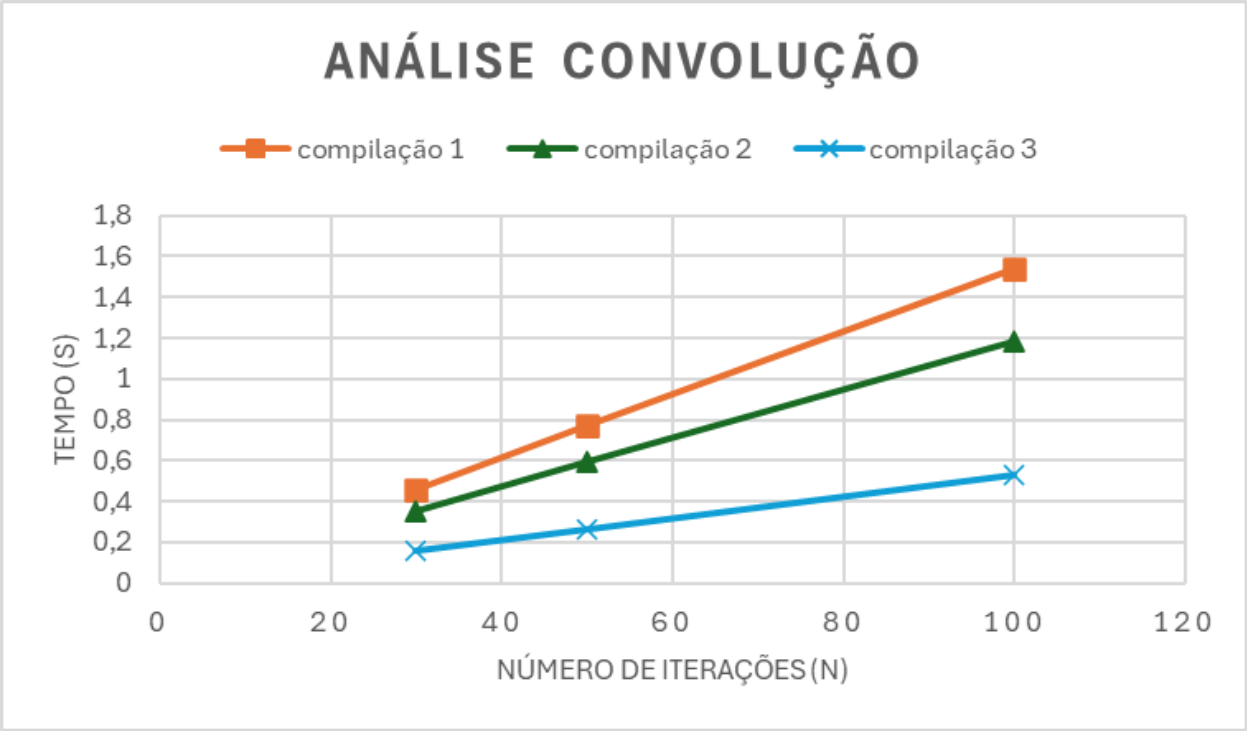
Relatório 5- Bruna Meinberg

Análise dos tempos das diferentes compilações

Utilizando os diferentes tipos de compilação para o código de convolução, encontrei o seguinte tempo:



Visivelmente, o basicão tem o desempenho pior, logo, utilizando somente os dados das compilações “diferentes”.



Utilizando a otimização avançada, é perceptível que o tempo é mais curto.

gprof e Callgrind

Compilando utilizando o gprof, o tempo levado para as 100 iterações foi o mesmo tempo da otimização básica. Já o callgrind não consegui rodar.

gprof

Analisando a “saída” do gprof, observamos o seguinte:

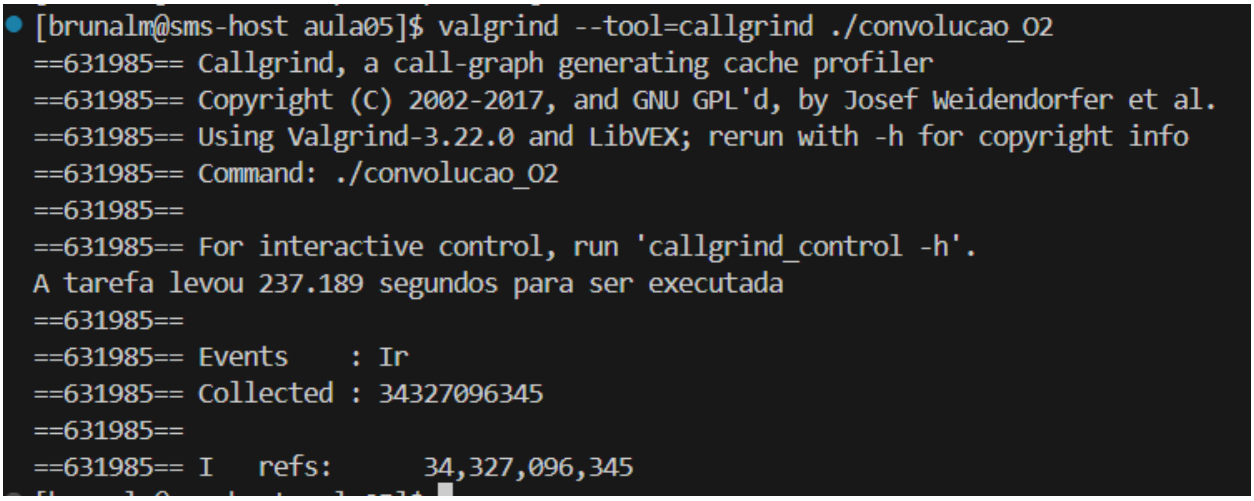
Each sample counts as 0.01 seconds.

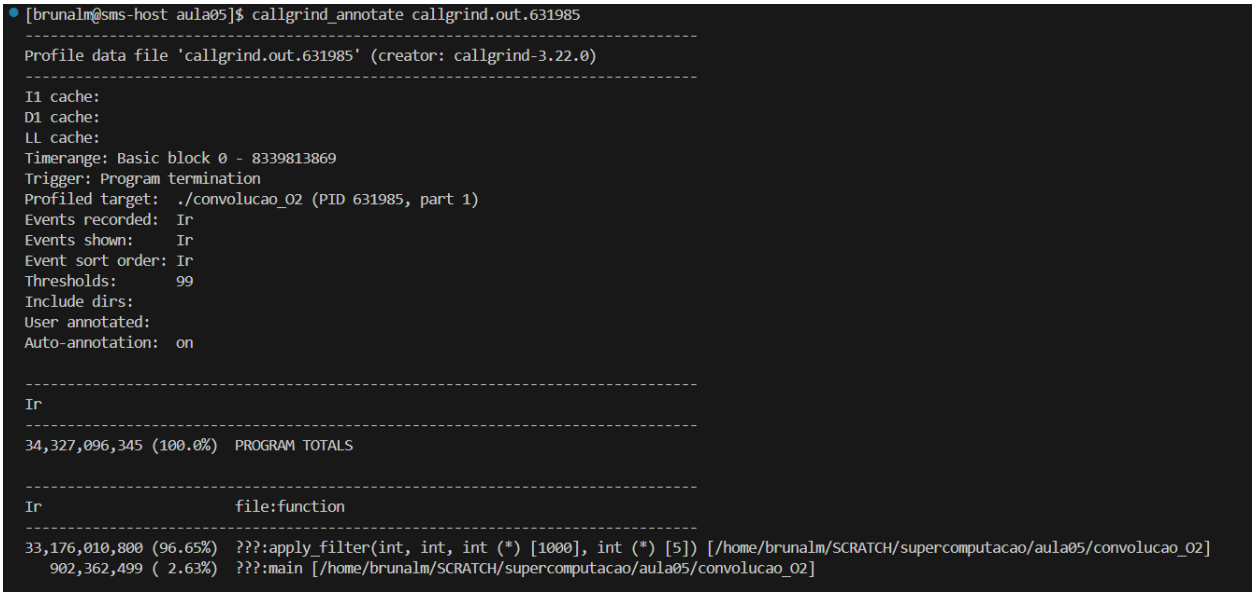
% time	cumulative seconds	self seconds	calls	self ns/call	total ns/call	name
91.01	0.81	0.81	100000000	8.10	8.10	apply_filter(int, int, int () *[1000]*, *int ()* [5])
6.74	0.87	0.06				main
2.25	0.89	0.02				_init

A maior tempo de código é gasto na chamada da função “apply_filter”, que é chamada muitas vezes.

Callgrind

Rodando os comandos:





Com isso vemos que a função apply_filter toma 96,65% do tempo, dando indicações de onde podemos otimizar.

Sugestões de otimização

De acordo com os resultados, seria interessante agir com otimizações em cima da função apply_filter, que com certeza é a mais chamada.

```
int apply_filter(int x, int y, int matrix[N][N], int filter[FIL
    int result = 0;
    int filter_offset = FILTER_SIZE / 2;  // Calcula o deslocame

    // Aplica o filtro 5x5 ao elemento (x, y) da matriz.
    for (int i = -filter_offset; i <= filter_offset; i++) {
        for (int j = -filter_offset; j <= filter_offset; j++) {
            int xi = x + i;
            int yj = y + j;
            // Verifica se o índice está dentro dos limites da m
            if (xi >= 0 && xi < N && yj >= 0 && yj < N) {
                result += matrix[xi][yj] * filter[i + filter_off
            }
        }
    }
    return result;  // Retorna o valor convoluído.
}
```

Provavelmente, devido ao tamanho da matriz (1000x1000 no caso dessa compilação), essa função acaba sendo chamada muitas vezes e, contendo dois loops alinhados, acaba tomando muito tempo.