Relatório: Trabalho 2 – Otimização de Desempenho

Gabriel Lisboa Conegero – GRR20221255
Pedro Folloni Pesserl – GRR20220072
Departamento de Informática
Universidade Federal do Paraná – UFPR
Curitiba, Brasil
glc22@inf.ufpr.br, pfp22@inf.ufpr.br

Resumo

Este relatório documenta o processo de otimização de um programa que realiza ajuste polinomial de curvas, utilizando o método dos **Mínimos Quadrados** e **Eliminação de Gauss**. Também apresenta a comparação entre as duas versões do programa, obtida a partir da ferramenta LIKWID.

1 Metodologia da análise

A análise do programa de ajuste polinomial de curvas foi feita considerando três seções principais do código, que realizam, respectivamente:

1. Geração do sistema linear pelo método dos Mínimos Quadrados;
2. Solução do sistema linear pelo método da Eliminação de Gauss;
3. Cálculo de resíduos do polinômio encontrado.

Tanto a seção de geração do sistema linear quanto a de cálculo dos resíduos do polinômio foram avaliadas com as seguintes métricas: tempo de execução, número de operações aritméticas de ponto flutuante por segundo (FLOP/s), com e sem uso de SIMD, banda de memória utilizada e taxa de *miss* na *cache* de dados. A seção de solução do sistema linear teve seu desempenho avaliado em tempo de execução e FLOP/s, apenas.

2 Otimizações realizadas

2.1 Geração do sistema linear

Originalmente, a versão 1 do programa utilizava uma tabela de *lookup* para armazenar as potências, de 0 a $2m$, dos pontos de entrada, onde m é o grau do polinômio a ser ajustado. Porém, isso precisou ser modificado, porque a entrada agora pode conter até 10^8 pontos, o que impossibilita o armazenamento das potências de todos os pontos na memória. Então, a versão 1 agora utiliza a função `pow_inter()` para calcular as potências dos x's a cada iteração.

A otimização empregada na versão 2 foi inverter a ordem dos laços no cálculo das potências, de forma que iteramos sobre o vetor de pontos de entrada apenas uma vez, possibilitando que ele seja mantido (ainda que em parte) em *cache*. Dessa maneira, percorremos a primeira linha e última coluna da matriz do sistema linear e o vetor de termos independentes múltiplas vezes, calculando a próxima potência do ponto atual que estamos somando a cada iteração com apenas uma multiplicação sobre a potência anterior.

Como o vetor de pontos é muito maior do que a matriz do sistema linear, que é sempre de ordem 5, espera-se que isso diminua a taxa de *miss* na *cache* por ser necessário recarregá-lo em *cache* menos vezes.

2.2 Cálculo de resíduos

Pela mesma questão apresentada na subseção 2.1, a versão 1 do programa foi modificada para usar a função `pow_inter()` na computação das potências dos pontos de entrada, para que fossem aplicadas no cálculo do polinômio com os coeficientes encontrados.

A versão 2 utiliza a mesma estratégia de multiplicar iterativamente cada x_i pela potência já calculada na iteração anterior, de forma a substituir a função custosa `pow_inter()` por uma multiplicação a cada passo.

2.3 Outras

Além do que já foi mencionado, demais otimizações incluem a substituição das funções de manipulação de intervalos por suas contrapartes `inline`, o que possibilita redução na taxa de *miss* na *cache* de instruções por não ser necessário desviar o fluxo de execução para a área de definição das funções.

3 Gráficos

Os gráficos aqui apresentados foram gerados por meio da ferramenta gnuplot.

3.1 Geração do sistema linear

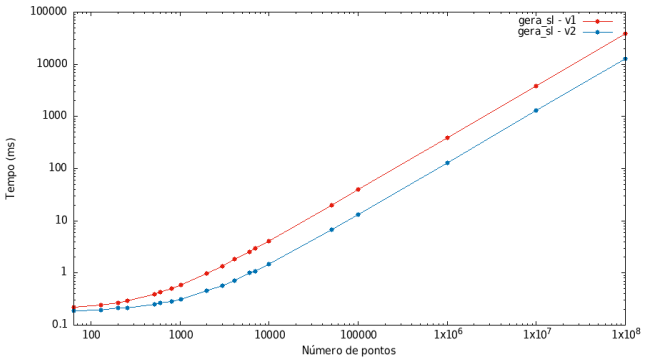

Figura 1: Tempo de execução.

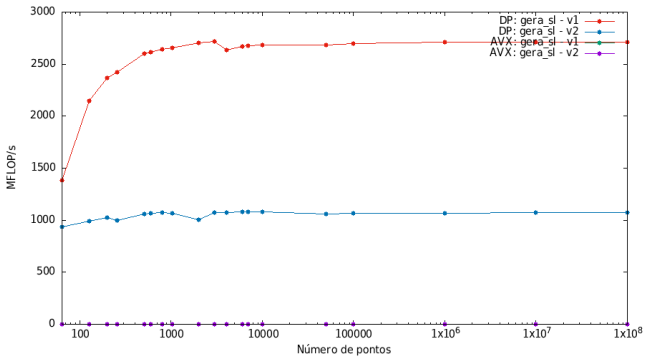

Figura 2: Número de operações de ponto flutuante por segundo.

Na figura 1 o tempo da versão 2 é aproximadamente 4 vezes menor. Devido as otimizações de uso da *cache* e cálculo da potência do intervalo.

Na figura 2 a quantidade de FLOP/s é muito maior na versão 1 devido ao uso da função `pow_inter()`. A versão 2 tem menos FLOP/s devido ao reaproveitamento dos cálculos para exponenciação do intervalo.

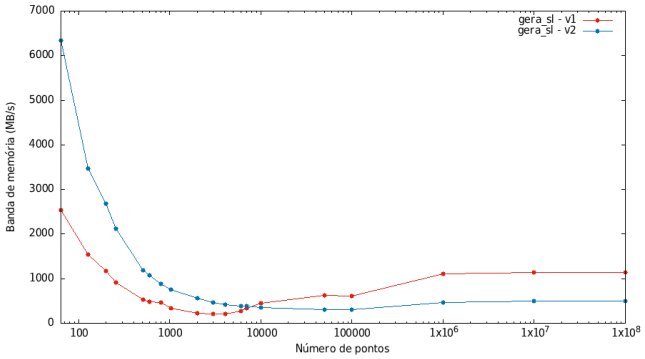

Figura 3: Banda de memória utilizada.

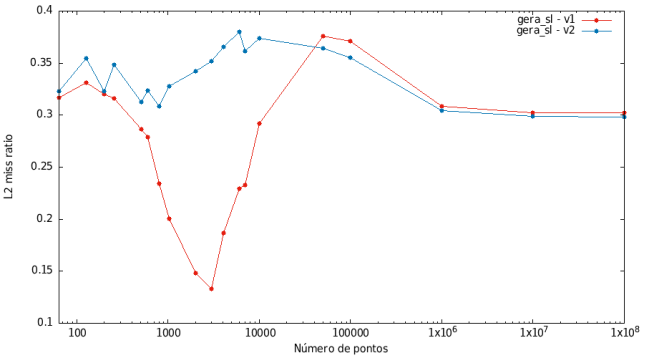

Figura 4: Taxa de *miss* na *cache* L2.

Na figura 3 a banda de memória é bem maior para um número pequeno de pontos devido a otimização que mantem os dados na *cache*. Porém conforme o número de pontos vai aumentando as taxas vão diminuindo e se igualando, devido ao aumento de *cache miss*.

Na figura 4 o comportamento foi inesperado, devido as otimizações de uso da *cache* o resultado esperado é que a versão 2 tenha um *cache miss* menor que a versão 1.

3.2 Solução do sistema linear

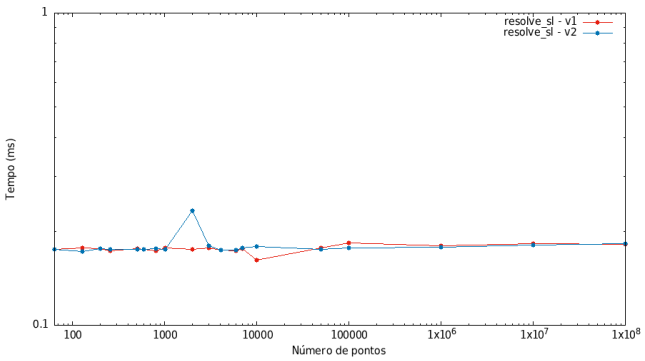

Figura 5: Tempo de execução.

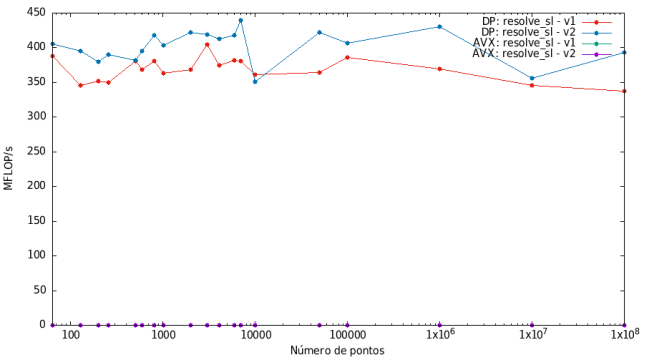

Figura 6: Número de operações de ponto flutuante por segundo.

Percebe-se pela figura 5 que o tempo de execução da solução do sistema linear foi aproximadamente constante, indepentendemente do número de pontos da entrada. Isso é devido ao fato de que a matriz sempre possui ordem 5.

A figura 6 apresenta uma constância independente do número de pontos e da versão, isso pois a matriz sempre possui ordem 5 e os cálculos não utilizam a função `pow_inter()`.

3.3 Cálculo de resíduos

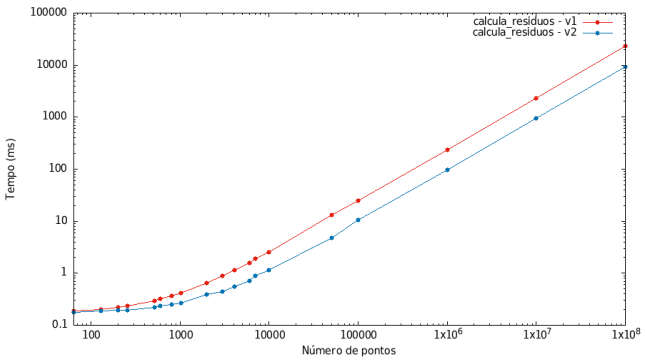

Figura 7: Tempo de execução.

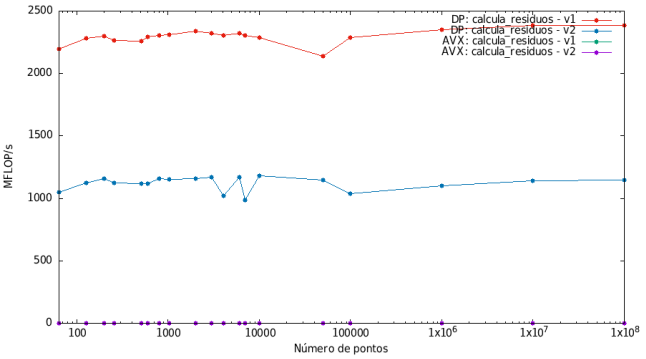

Figura 8: Número de operações de ponto flutuante por segundo.

A figura 7 apresenta que o tempo de calcular os residuos é aproximadamente 4 vezes maior na versão 1 do que na versão 2, devido à otimização do uso da função `pow_inter()`.

O número de FLOP/s é muito menor na versão 2, como pode ser visto na figura 8. Isso pois a versão 2 otimiza o código e reaproeita a potência anterior de x_i para calcular a nova potência, ao invés de utilizar a função `pow_inter()`, que é custosa.

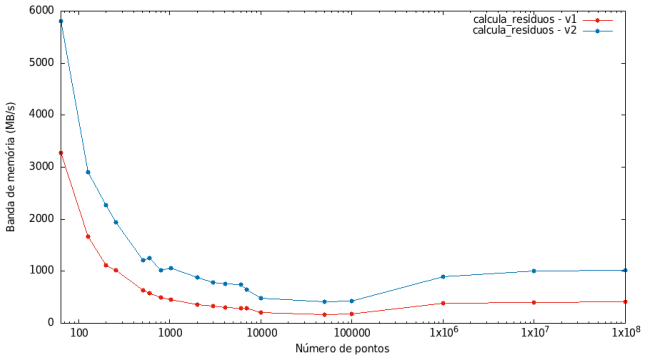

Figura 9: Banda de memória utilizada.

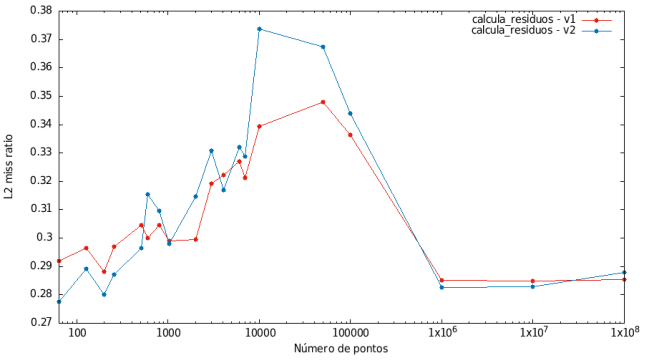

Figura 10: Taxa de *miss* na *cache* L2.

Em ambas as figuras 10 e 10 os valores são parecidos nas duas versões, isso pois a otimização no cálculo de resíduos foi na parte de operações aritméticas.