Avaliação de Desempenho da Solução de uma Equação Diferencial Parcial Elíptica

Gustavo Barros de Alcântara[1], **Lucas Akihiko Osato Ikeda**[1]

[1]Universidade Federal do Paraná (UFPR)
Curitiba – PR – Brazil

gba12@inf.ufpr.br, laoi12@inf.ufpr.br

1. Introdução

1.1. O código implementado

A primeira versão do código para a resolução da equação parcial elíptica, foi muito bem estruturado de forma a impedir que operações aritméticas "caras" fossem utilizadas muitas vezes dentro do loop principal das funções que calculam o método SOR e a função que calcula o resíduo. Todos os valores de seno que iriam ser utilizados na equação principal foram computados antes de se chegar ao loop, visto que estes valores são fixos para cada ponto da matriz a partir do momento que se sabe as dimensões desta última. As estruturas de dados usadas foram matrizes projetadas em vetores para prover maior integridade e localidade espacial da memória alocada, diminuindo assim a quantidade de cache miss.

2. Análises

Foram computadas as análises do software pdeSolver com o auxílio de um script escrito em bash, desenvolvido pelos próprios autores, de forma a permitir recalcular os testes de maneira dinâmica a cada nova alteração do código e gerar os novos dados para a plotagem dos gráficos. Os testes realizados foram feitos nos aspectos de: Tempo, banda de memória, cache miss e operações aritméticas.

2.1. Otimização do código

A otimização realizada no código foram diferentes para cada função, no cálculo do SOR e do resíduo foi reduzido o número de operações redundantes e foram retiradas as variáveis auxiliares, pois assim não era necessário esperar os valores serem atualizados nos registradoes, o que pode causar "bolhas" no pipeline, e assim ganhar vários ciclos de clock em cada iteração. No SOR o método de "Loop Unrolling" foi utilizado para reultilizar os valores buscados pelo pre-fetch e assim diminuir cache miss e aumentar a velocidade.

2.2. Tempo

Observando o gráfico da análise do tempo na figura 1 a seguir, pode-se observar que o cálculo do resíduo em v2 foi bem otimizado e está muito mais rápido que sua contrapartida em v1. O mesmo ocorre para o SOR que ganhou desempenho com o loop unrolling. Porém por ter mais dependencia de dados em memória (memory-bound), o SOR de v2 não ficou tão bom quanto o resíduo de v2.

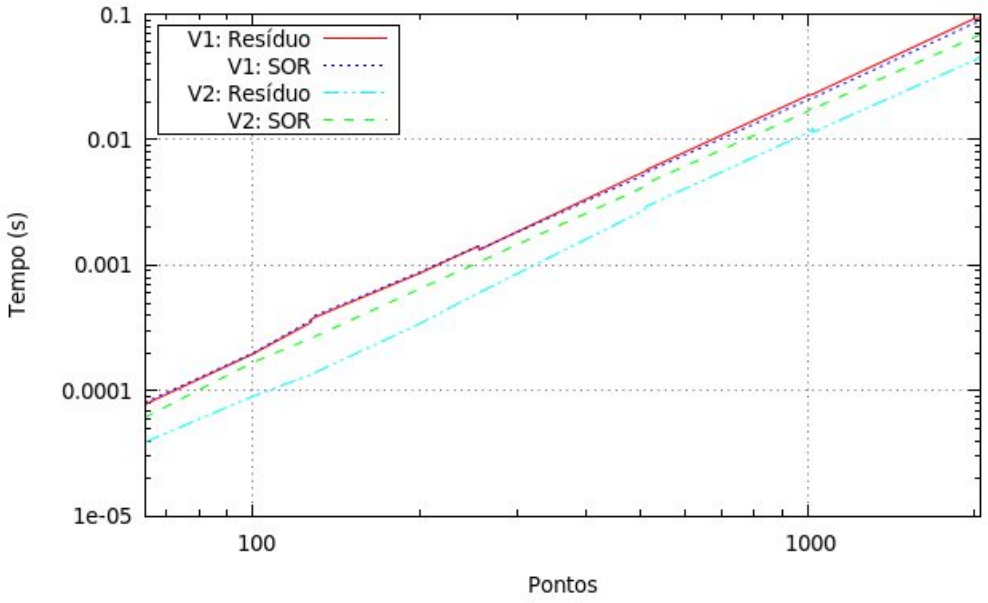

Figura 1. Análise de Tempo

2.3. Banda de Memória

A análise da banda de memória está intimamente ligada à análise de cache miss. Conforme a capacidade de memória das caches acabam, a banda de memória começa a crescer de forma acentuada até estabilizar no limite do barramento. Tal comportamento é ilustrado na figura 2 a seguir.

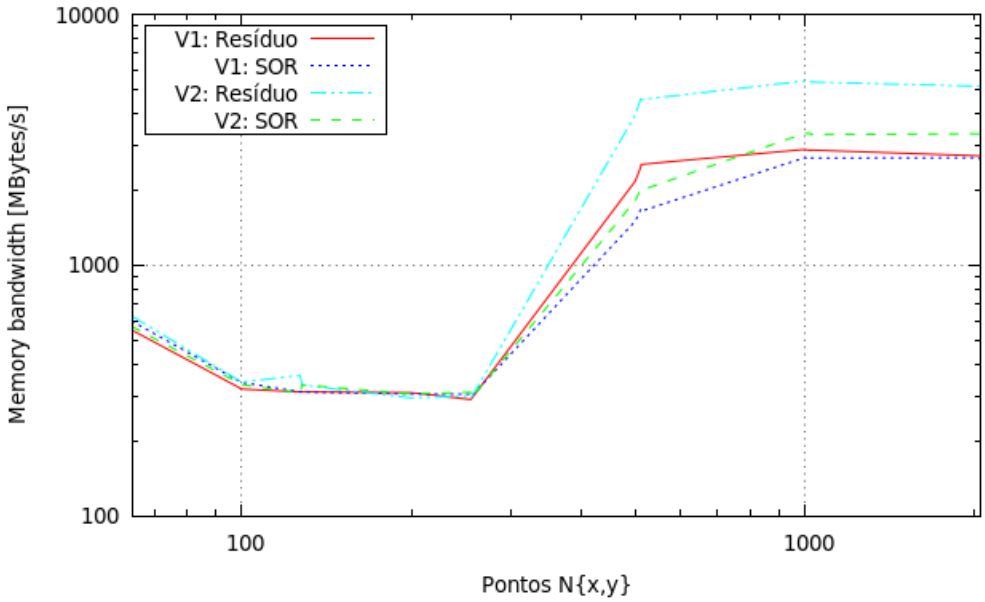

Figura 2. Análise de Memória

2.4. Cache Miss

O cache miss do resíduo em v2 aumentou bastante, isso se deve por ele ter que acessar uma matriz a mais para realizar os cálculos. O SOR em v2 teve diminuição de cache miss, porém o maior número de acessos à memória em seus vetores para realizar "store" impede que tenha uma diminuição ainda maior. Podemos observar no gráfico ilustrado na figura 3, a seguir, que o data cache miss ratio salta em alguns pontos, isso se deve ao motivo do número de elementos na matriz ser múltiplo do tamanho da cache, que acaba ocasionando um "cache trashing".

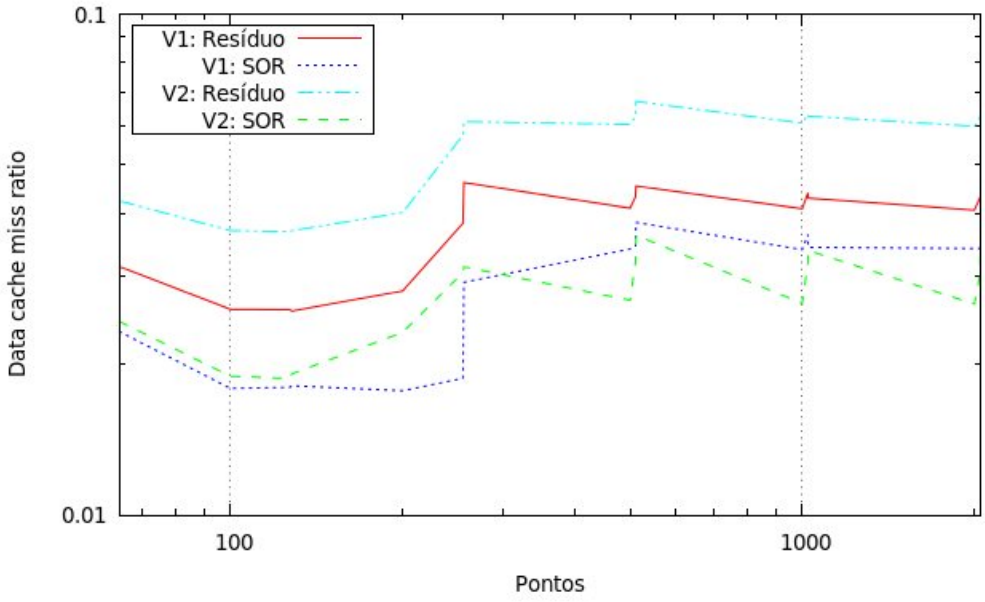

Figura 3. Análise de Cache

2.5. Operações aritméticas

O SOR faz mais FLOPS que o resíduo pois ele tem menos cache miss comparado ao resíduo, logo o resíduo deve esperar mais tempos por dados, o que faz com que ele não possa acompanhar o FLOPS do SOR.
Como o SOR da v2 possue menos cache miss, é observável que ele faz mais FLOPS que a sua versão v1.

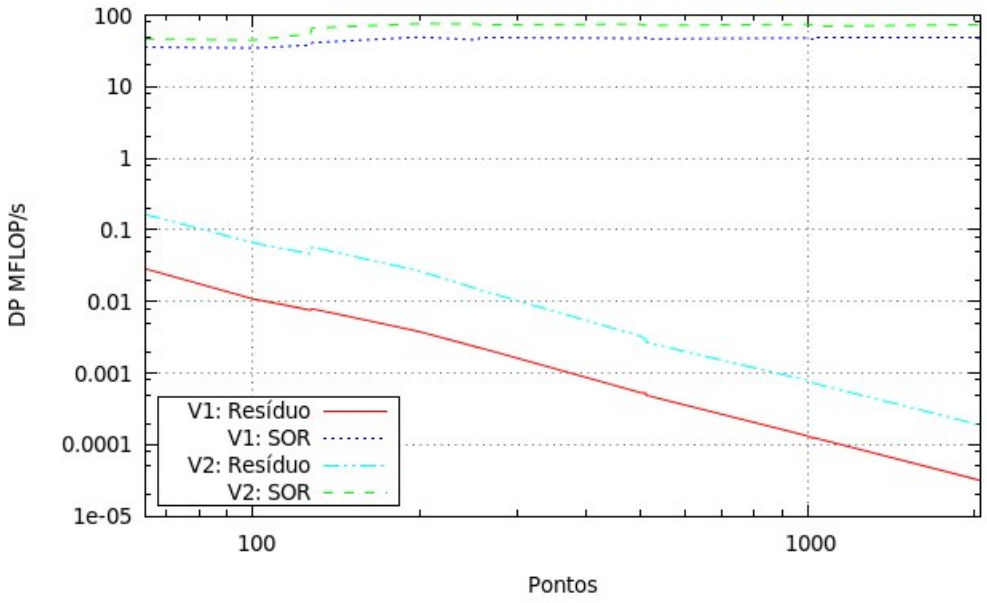

Figura 4. Análise de operações aritméticas

3. Conclusão

Como a v1 estava bem otimizada, onde já estava sendo feito vários cálculos fora da parte crítica para economizar tempo de processamento e a matriz estava estruturada como vetor para uma melhor localidade espacial e, assim, melhorar o tempo de acesso, o v2 deu uma polida melhor nesses aspectos para melhorar ainda mais.
Também foi feito algumas melhoras no acesso da matriz para facilitar o uso do pre-fetch e diminuir o gasto de ciclos de clock acessando memória, como por exemplo o loop unroll e a retirada de variáveis desnecessárias. Foi observado que o loop unroll causou dependencias entre os elementos da matriz, pois o cálculo do atual necessitava do resultado do anterior, gerando bolhas no pipeline. Isso foi contornado ao se adicionar variáveis temporárias para guardar os valores já calculados.
Também foi observado que ocorre "cache trashing" em alguns pontos, já que estes são multiplos do tamanho da cache. Porém os problemas encontrados ao tentar contornar este erro foram maiores do que se era esperado.