Estruturas de Dados - Aula Prática 06 - Relatório

Lucas de Oliveira Araújo, 2022036039

<2023-05-15 Mon>

Contents

---

1 Análise qualitativa

Para a realização dessa tarefa, escolhi o resolver de expressão numérica implementado por mim para o trabalho prático 1.

O programa em questão possivelmente terá acessos à memória em maior quantidade durante a execução das funções *postfixIsValid*, *infixIsValid*, *storeExpression* e *evaluation*. Isso se dá pelo falo de tais funções manipularem a expressão armazenada nas estruturas de dados do programa.

Em meu programa é esperado um nível expressivo de localidade de referência temporal, uma vez que tanto as funções utilizadas para verificar a validade da expressão quanto as funções que à manipulam na árvore binária acessam o mesmo dado várias vezes. Por exemplo, para a verificação da validade de uma expressão, esta é verificada manipulando uma string, ou seja, ao acessar o primeiro elemento da string, aumenta a probabilidade de acessar os posteriores. Além disso, na manipulação da expressão já armazenada na árvore, acessar o nó raiz aumenta a probabilidade de acessar os nós filhos.

Já em relação a localidade de referência espacial, devido a todas as estruturas de dados do meu programa utilizarem nós com ponteiros que os interligam e devido a forma como os dados são armazenados no heap (array contíguo), é presumível que ela seja relativamente alta.

2 Plano de caracterização

Para a análise, vou expor o programa a uma expressão e utilizarei os comandos internos do programa para controlar a manipulação dessa expressão em diversos cenários, ou seja, farei a análise de como cada função do programa acessa e manipula a memória, de forma a encontrar o comportamento padrão de cada uma delas.

A sequência de comandos inseridas no arquivo "commands.txt" foi a seguinte:

```
LER <expressão de 1000 tokens>
INFIXA
POSFIXA
RESOLVE
LER <expressão de 3 tokens>
INFIXA
POSFIXA
RESOLVE
LER <expressão de 1000 tokens>
INFIXA
POSFIXA
RESOLVE
LER <expressão de 1000 tokens>
INFIXA
POSFIXA
RESOLVE
```

O intuito de inserir mais de uma expressão é possibilitar analisar também o destrutor da árvore binária que armazenava a expressão anterior.

Os experimento serão conduzidos utilizando a ferramente valgrind com as opções:

1. –tool=cachegrind: para verificar o comportamento dos caches durante a execução;
2. –tool=callgrind: para verificar os recursos utilizados durante a execução.

3 Avaliação dos experimentos

3.1 Análise do resumo do cachegrind

Após executar os procedimentos citados no plano de caracterização, observamos que, em geral, o programa apresentava uma baixa taxa de *misses* de leitura nos cache L1 e L2, conforme figura 1.

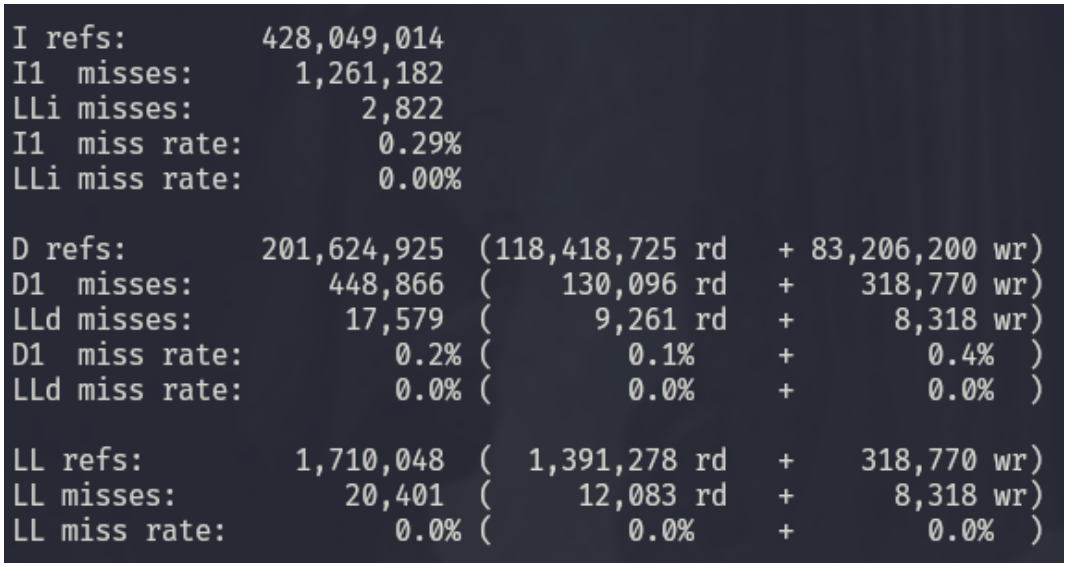

Figure 1: Resumo da avaliação do cachegrind

Com relação ao número de operações realizadas (I refs), a taxa de *misses* de instruções no cache L1 (l1 miss rate) foi de 0.29%, o que indica que a maioria das instruções foram encontradas nesse cache. Além disso, a taxa de *misses* de dados no cache L1 (D1 miss rate) foi de apenas 0.2%, também consideravelmente baixa, o que significa que a maioria dos dados buscados foram encontrados nesse cache.

Já com relação ao cache L2, o nível de misses de dados (LLd miss rate) foi tão baixa que a precisam de saída do valgrind não pôde representa-la. Uma possível explicação para essa pequena taxa é que o tamanho do cache L2 é suficiente para armazenar os dados do programa e, portanto, não houve erros de leitura. A taxa de misses na cache L2 de instruções (LLi miss rate) também é muito baixa, de 0.0%. Isso significa que a maioria das instruções foram encontradas na cache de nível L2.

Por fim, o número total de referências à cache de nível L2 (LL refs) é de 1,710,048. Comparado com o número de instruções, o total de referências é bem baixo, o que pode ser explicado pelo baixo volume de dados que o programa lidou (expressões de no máximo 1000 tokens).

3.2 Análise do desempenho dos segmentos de código do programa

Conforme o esperado, as funções de conversão entre a notação posfixa e infixa tiveram um número razoável instruções realizadas, além de apresentar um número baixo de misses, conforme figura 2.

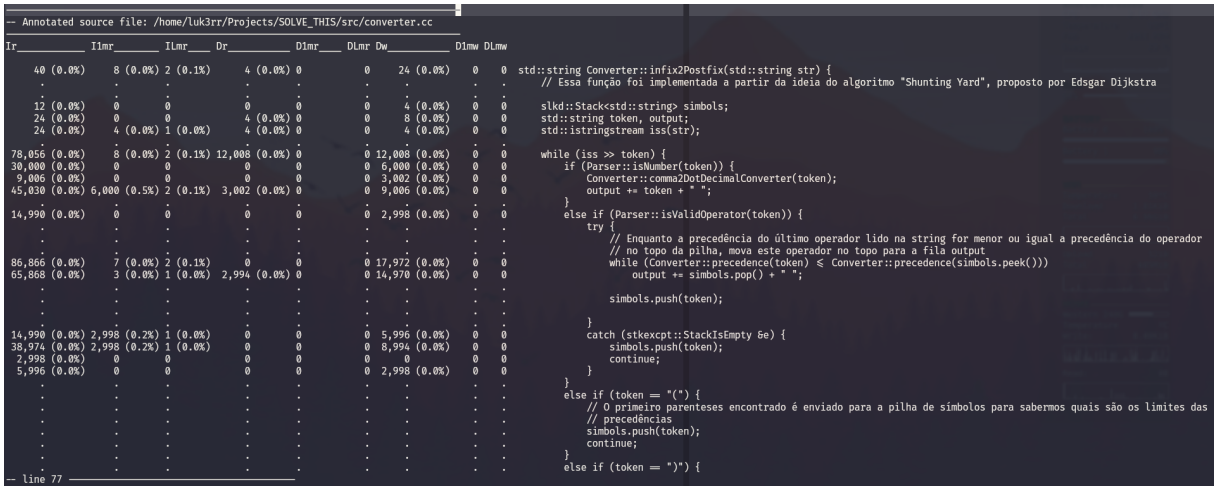

Figure 2: Conversão de infixa para posfixa

De todas as funções do programa, a que teve maior número de instruções realizadas foi a função de caminhamento pos-ordem da árvore binário, conforme figura 3. Isso pode ser explicado pelo fato de se tratar de uma função recursiva. Além do mais, houve um número considerável de misses de gravação no cache L1.

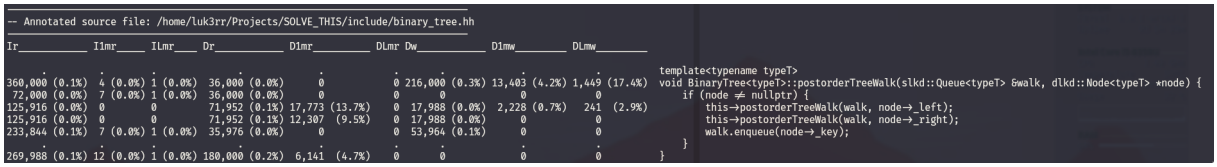

Figure 3: Caminhamento pós-ordem

Em geral, as funções do programa tiveram misses de leitura e gravação abaixo de 0.5% em ambos os caches, o que pode ser um bom indicativo de performance.

4 Conclusão

Pela análise realizada, podemos concluir que o programa apresenta alta localidade de referência, uma vez que o miss rate é baixo e, portanto, o cache dos níveis L1 e L2 estão sendo bem aproveitados. Por outro lado, em certas partes do código é possível notar um número elevado de instruções, o que pode ser um indicativo de uma implementação não eficiente.

Ademais, como o miss rate aumenta do cache L1 para o L2, podemos concluir que a localidade de referência do programa é predominantemente espacial, pois a mesma localidade de dados não é acessada várias vezes em um curto período de tempo, mas sim em diferentes momentos do programa, fazendo com que os dados sejam frequentemente trocados no cache.