Aula Prática 6

Lucas Albano Olive Cruz - 2022036209

1. Avaliação Qualitativa

O programa recebe uma expressão e a armazena em uma árvore, antes disso ele executa uma série de verificações. Para cada verificação são feitas diversas comparações e na fase de inserção diversas escritas. Ainda, são feitas diversas chamadas recursivas ao caminhar pela árvore e resolver a expressão. Estas partes devem ser as mais custosas do programa.

Plano de caracterização Localidade Referência

Na fase de verificação várias sequências de localidades provavelmente serão acessadas pois a expressão é lida de forma contígua. Após isso haverá uma transição para a parte de inserção na quais partes não necessariamente contíguas da memória serão acessadas pois a árvore é montada de forma dinâmica.

3. Parâmetros Caracterizados

Portanto, serão analisados os parâmetros de acessos de memória (leitura e escrita) nas fases de validação, inserção, caminhamento e resolução da expressão na árvore sintática. Além do número de chamadas das principais funções do programa.

Será usada uma entrada de tamanho médio seguida de operações de conversão e resolução.

4. Execução com Cachegrind

```
Profile data file 'cachegrind.out.5212'

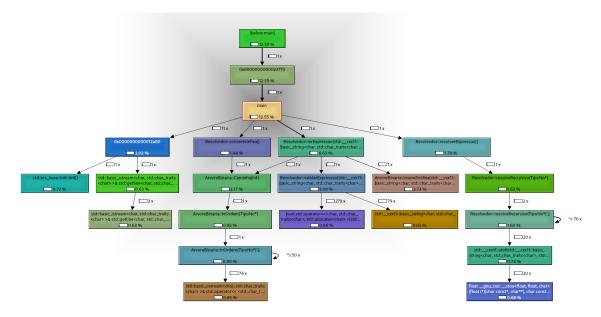
Il cache: 32768 B, 64 B, 8-way associative
Dl cache: 32768 B, 64 B, 8-way associative
LL cache: 2097152 B, 64 B, 8-way associative
Profiled target: ./bin/programa
Events recorded: Ir Ilmr ILmr Dr Dlmr DLmr Dw Dlmw
Events shown: Ir Ilmr ILmr Dr Dlmr DLmr Dw Dlmw DLmw
Event sort order: Ir Ilmr ILmr Dr Dlmr DLmr Dw Dlmw DLmw
Thresholds: 99 0 0 0 0 0 0 0 0
Include dirs:
User annotated:
Auto-annotation: on
```

Ir	11	lmr	ILmr	Dr	D1mr	DLmr	Dw
	D1 mw	DT.mw					

3,024,453 (100.0%) 2,538 (100.0%) 2,255 (100.0%) 786,430 (100.0%) 19,551 (100.0%) 9,063 (100.0%) 275,329 (100.0%) 2,600 (100.0%) 1,587 (100.0%) PROGRAM TOTALS

A partir daqui temos uma saída extensa referente aos acessos de memória de cada função. Para não poluir o relatório irei deixá-las de fora.

5. Execução com Callgrind



6. Avaliação das Saídas

Com o cachegrind conseguimos extrair algumas informações como número total de instruções realizadas, número de leituras na cache L1, número de escritas na cache L1 e perdas.

Assim, percebemos que o programa executa um total de 3,024,453 instruções, 786,430 leituras e 275,329 escritas.

As nossas principais funções tiveram os seguintes resultados:

Função	Instruções	Leitura	Escrita
validarExpressao	24.987	7.044	2.680
inserePosfixa	11.650	2.489	1.556
resolveRecursivo	3.993	1.331	786
inOrdem	2.038	510	471
posOrdem	1.648	510	393

Assim é evidente que a fase de validação da expressão é altamente custosa por necessitar de várias instruções de leituras para comparar os caracteres da expressão. Ainda é interessante notar que ambos métodos de caminhamento gastam a mesma quantidade de leituras pois percorrem a mesma árvore sintática, no entanto o caminhamento inOrdem gasta mais escritas por necessitar de adicionar parênteses.

Agora analizando o callgrind também possuímos algumas informações interessantes. A função validaExpressao faz 358 chamadas a funções std de comparação, novamente, percebemos ser um método muito custoso ao programa. O caminhamento inOrdem faz 50 chamadas recursivas a ele mesmo e 74 chamadas ao ostream para escrever no terminal. O método resolveRecursivo faz 76 chamadas recursivas e 20 chamadas a função stof para converter uma string para um float.

Assim, creio que exceto a fase de validação, o programa se comporta relativamente bem em relação aos gastos de memória. A estrutura de dados árvore é bastante eficiente para armazenar e possibilitar acesso à expressão.