Estruturas de Dados

Aula Prática 2

Nome: Letícia Scofield Lenzoni
Matrícula: 2022035547

Tarefa 1: Cálculo de Fatorial

Implementações:

- Fatorial Iterativo (fatorialIterativo.cpp)
- Fatorial Recursivo (fatorialRecursivo.cpp)

Experimentos Realizados:

Para os códigos de fatorial, a faixa de valores que serão avaliadas é de 1 a 15, sendo analisados os valores 1, 5, 10 e 15.

Resultados:

	Versão Recursiva			
	Fatorial de 1	Fatorial de 5	Fatorial de 10	Fatorial de 15
Tempo de relógio (em ns)	5082	3596	3508	2448
Tempo de usuário (em ns)	700	1600	1200	1000
Tempo do sistema (em ns)	393	480	714	650

	Versão Iterativa			
	Fatorial de 1	Fatorial de 5	Fatorial de 10	Fatorial de 15
Tempo de relógio (em ns)	3641	3879	3006	3744
Tempo de usuário (em ns)	1200	1800	1100	1300
Tempo do sistema (em ns)	507	651	521	722

Análise:

- A versão recursiva do cálculo de fatorial mostrou um aumento consistente no tempo de execução à medida que o número (n) aumentou, de acordo com a complexidade O(n).
- A versão iterativa teve tempos de execução mais consistentes e flutuações menores, indicando maior estabilidade.

- Comparando ambas as versões, a versão iterativa demonstrou melhor desempenho em termos de tempo de execução para valores maiores de n.
- A versão recursiva mostrou maior variação nos tempos de usuário e sistema, sugerindo maior sobrecarga de sistema associada à recursão.
- A escolha entre as versões depende dos requisitos do projeto, com a versão iterativa sendo mais eficiente em termos de desempenho, enquanto a versão recursiva pode ser preferível em termos de clareza e simplicidade do código.

Gprof:

Os resultados do profiling com o Gprof para diferentes valores de "n" (1, 5, 10 e 15) indicam que a função "Fat(int)" é a função que consome a maior parte do tempo de execução do programa em todos os casos. A função "__static_initialization_and_destruction_0(int, int)" também é listada, mas não contribui significativamente para o tempo de execução.

- Para n = 1: A função "Fat(int)" foi chamada uma vez e não levou tempo significativo.
- Para n = 5: A função "Fat(int)" foi chamada cinco vezes, mas também não consumiu tempo significativo.
- Para n = 10: A função "Fat(int)" foi chamada 55 vezes, mas ainda não consumiu tempo significativo.
- Para n = 15: A função "Fat(int)" foi chamada 610 vezes, mas o tempo gasto ainda é insignificante.

Com adição de seno:

O tempo gasto ainda foi insignificante, porém, aumentou drasticamente em comparação com o teste de fatorial anterior. Sendo o último relatório:

time	seconds	seconds	calls	Ts/call	Ts/call	name
100.00	14.71	14.71				Fat(int)
0.00	14.71	0.00	1219	0.00	0.00	__do_global_dtors_aux
0.00	14.71	0.00	1	0.00	0.00	main

Tarefa 2: Número de Fibonacci

Implementações:

- Fibonacci Iterativo (fibonacciIterativo.cpp)
- Fibonacci Recursivo (fibonacciRecursivo.cpp)

Experimentos Realizados:

Para os códigos de fibonacci, a faixa de valores que serão avaliadas é de 1 a 15, sendo analisados os valores 1, 5, 10 e 15.

Resultados:

	Versão Recursiva			
	Fibonacci 1	Fibonacci 5	Fibonacci 10	Fibonacci 15
Tempo de relógio (em ns)	3967	2671	2410	7241
Tempo de usuário (em ns)	1200	1100	1100	5700
Tempo do sistema (em ns)	319	687	823	4940

	Versão Iterativa			
	Fibonacci 1	Fibonacci 5	Fibonacci 10	Fibonacci 15
Tempo de relógio (em ns)	2744	3700	2890	6371
Tempo de usuário (em ns)	1000	1400	1600	5200
Tempo do sistema (em ns)	436	645	1104	4820

Análise:

- A versão recursiva do cálculo de Fibonacci também mostrou um aumento consistente no tempo de execução à medida que o valor de Fibonacci (n) aumentou, seguindo uma complexidade O(2^n), que é exponencial.
- A versão iterativa exibiu tempos de execução mais consistentes e flutuações menores em comparação com a versão recursiva, o que indica maior estabilidade no desempenho.
- Comparando ambas as versões, a versão iterativa continuou a demonstrar um desempenho superior em termos de tempo de execução, especialmente para valores maiores de n. Isso é consistente com a análise anterior para o cálculo de fatorial.
- Assim como na análise anterior, a versão recursiva mostrou maior variação nos tempos de usuário e sistema, sugerindo uma sobrecarga de sistema associada à recursão.

Gprof:

Os resultados da análise de perfil com o Gprof para diferentes valores de "n" (1, 5, 10 e 15) indicam que a função "Fib(int)" é a função que consome a maior parte do tempo de execução do programa em todos os casos. A função "__static_initialization_and_destruction_0(int, int)" também é listada, mas não contribui significativamente para o tempo de execução.

- Para n = 1: A função "Fib(int)" foi chamada uma vez e não levou tempo significativo.
- Para n = 5: A função "Fib(int)" foi chamada oito vezes, mas também não consumiu tempo significativo.
- Para n = 10: A função "Fib(int)" foi chamada 1.218 vezes, mas ainda não consumiu tempo significativo.
- Para n = 15: A função "Fib(int)" foi chamada 14.714 vezes, mas o tempo gasto ainda é insignificante.

Com adição de seno:

O tempo gasto ainda foi insignificante, porém, aumentou drasticamente em comparação com o teste de fibonacci anterior. Sendo o último relatório:

time	seconds	seconds	calls	Ts/call	Ts/call	name
100.00	14.71	14.71				Fat(int)
0.00	14.71	0.00	1219	0.00	0.00	__do_global_dtors_aux
0.00	14.71	0.00	1	0.00	0.00	main