

Relatório do Trabalho 2 de Sistemas Operacionais

Romeu Tamarozi Bernardes

1. Programa em C++ para cálculo da série de Leibniz

A primeira parte do trabalho consiste em fazer um programa em C++ que calcule uma aproximação de π a partir da série de Leibniz, da seguinte forma:

$$\pi \approx 4 \cdot \sum_{k=0}^{N-1} \frac{(-1)^k}{2k+1}$$

O código para esta etapa do trabalho está presente em *sequencial.cpp*, neste diretório.

Explicação do código (*sequencial.cpp*)

No meu código, a quantidade de termos da série que serão calculados é definida por input do usuário, suportando teoricamente até $2^{63} - 1$ termos, pois a variável que recebe o input é de tipo *long long*.

A variável ***N*** armazena a quantidade de termos da série de Leibniz que serão calculados, recebendo um input do usuário como valor. O laço de repetição *for*, que tem o contador ***k*** indo de 0 até ***N***, realiza o somatório que determina os termos da série, sendo o resultado do somatório armazenado em ***soma***.

Para a alternância de sinal dos termos ao longo da série, eu utilizei uma condição na definição de valor da variável ***sinal*** (que armazena o numerador para a divisão que gera os termos): se ***k*** for par, ***sinal*** será positivo; caso contrário, ***sinal*** será negativo. Esta condição faz o mesmo papel da expressão $(-1)^k$ na série, sendo mais simples do que a utilização de potência.

Finalizado o laço *for*, o resultado do somatório será multiplicado por 4 para obter a aproximação de π , na variável ***pi***, que logo após é impressa com 15 casas decimais.

2. Cálculo distribuído entre *threads*

Na segunda parte do trabalho, o objetivo é modificar o código anteriormente escrito para que o cálculo seja distribuído entre múltiplas tarefas, em múltiplas *threads*, utilizando *pthreads* no Linux. Também devem ser realizados testes de tempo de execução com o uso de 1, 2, 4 e 8 *threads*.

O código para esta etapa do trabalho está presente em *threads.cpp*, neste diretório.

Explicação do código (*threads.cpp*)

(Esta explicação será mais voltada à estrutura geral do programa. O código está detalhadamente comentado, então para explicações mais específicas sobre cada variável, função, etc. do código recomendo ler os comentários.)

Aqui, o cálculo da série de Leibniz é dividido entre múltiplas *threads*, que operam paralelamente. Cada *thread* calcula uma parcela dos termos da série e armazena seus resultados parciais em um vetor **resultado** compartilhado.

É definida uma estrutura de dados **ThreadData**, que define as informações necessárias para que cada *thread* execute sua porção do processo (intervalo de iteração, posição dentro do vetor compartilhado e referência ao vetor compartilhado).

Também é definida a função **thread_func** que cada *thread* executa. É aqui que a soma parcial de um intervalo da série é realizada.

A função principal (**main**) recebe dois argumentos da linha de comando: a quantidade de termos da série que serão calculados e a quantidade de *threads* que serão utilizadas. Então, ela aloca memória para as *threads*, as informações **ThreadData** e o vetor **resultado**, e cria e inicializa cada *thread* com sua respectiva posição no vetor compartilhado e seu bloco de termos para calcular (**BlocoDeIteracoes**). A execução do programa é então bloqueada até que as *threads* finalizem suas somas parciais e as armazenem no vetor **resultado**. Quando todos os cálculos parciais são finalizados eles são todos somados, e esta soma final é armazenada na variável **soma**, que é então multiplicada por 4 para obter-se a aproximação de π . O resultado final é impresso no terminal, a memória alocada é liberada e o programa é finalizado.

Execução do programa

No programa *threads.cpp*, a quantidade de termos calculados e o número de *threads* usadas devem ser passados como argumentos na execução do código compilado. Então, a forma de execução fica (para Linux): `./threads <quantidade_de_termos> <quantidade_de_threads>`

Comparação de tempos de execução

Para a comparação de tempos de execução para diferentes quantidades de *threads*, foi utilizado o comando *time* no Linux (formato de execução: `time ./threads <quantidade_de_termos> <quantidade_de_threads>`), e observado o tempo real passado por ele.

Foram realizados testes com 1, 2, 4 e 8 *threads*, todos calculando 10^9 termos da série de Leibniz. Para cada número de *threads* foram realizados cinco testes, e ao final foram medidas as médias de tempo para cada uma.

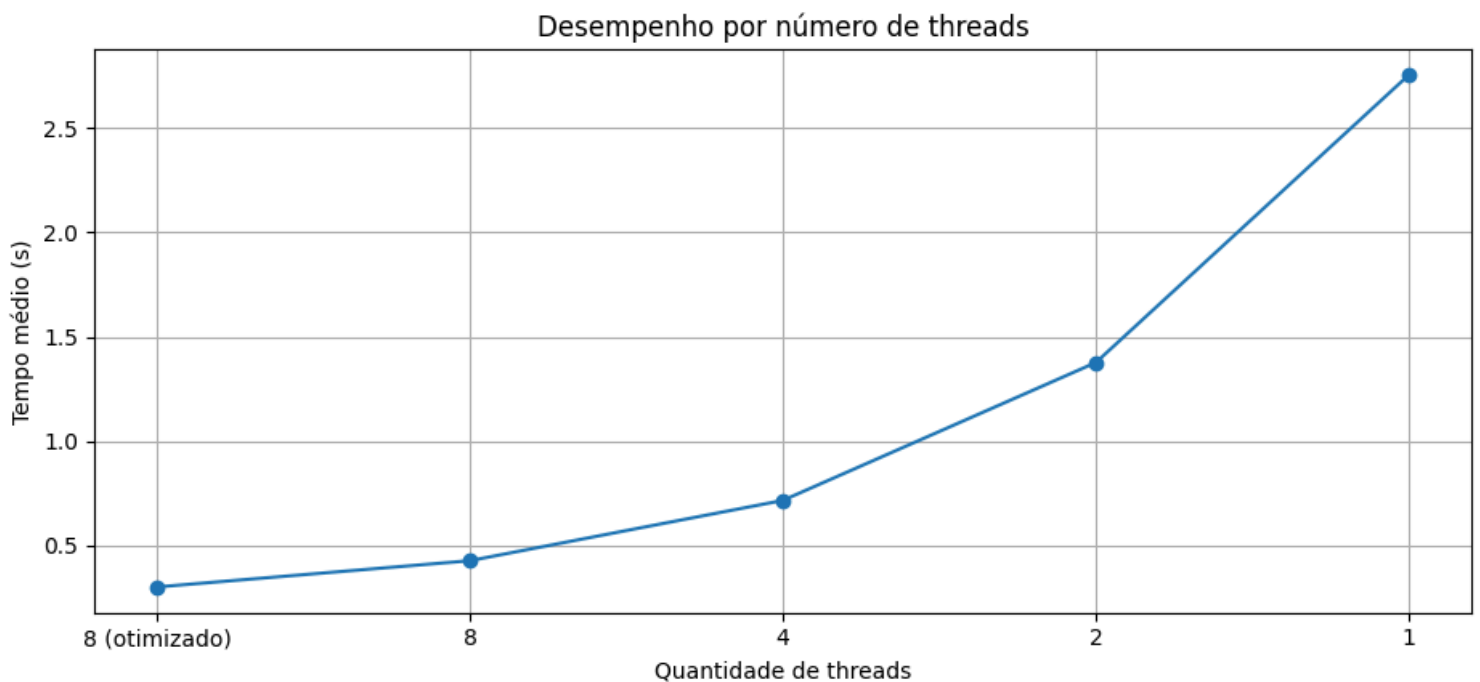
Eu também realizei, por curiosidade, mais cinco testes adicionais com otimização automática na compilação, utilizando o argumento `-O2` ao compilar o código por GCC. Estes testes foram realizados com 8 *threads* e 10^9 termos da série

de Leibniz, e os resultados serão incluídos junto com os testes obrigatórios para o trabalho.

Eis uma tabela com os resultados obtidos, em segundos:

Quantidade de threads utilizadas	Teste 1	Teste 2	Teste 3	Teste 4	Teste 5	Média
8	0,432	0,429	0,429	0,420	0,426	0,427
4	0,716	0,721	0,729	0,707	0,708	0,716
2	1,403	1,388	1,366	1,367	1,362	1,377
1	2,822	2,694	2,724	2,795	2,744	2,756
8 (otimizado)	0,290	0,292	0,320	0,304	0,299	0,301

E aqui um gráfico representando visualmente a diferença de velocidade de desempenho:



3. Referências utilizadas:

Publicações no site Open Group sobre a biblioteca *pthread*:
<https://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread.h.html>