UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
Departamento de Engenharia de Computação e Automação

Programação Paralela
Juliane da Silva Santos

Relatório 3:
Aproximação matemática

1 Introdução

O cálculo de aproximações numéricas de constantes matemáticas, como o valor de Π, é um problema clássico em ciência da computação e matemática. A série de Leibniz oferece uma maneira simples e intuitiva de calcular o valor de Π por meio de uma soma infinita de termos. No entanto, como qualquer aproximação numérica, ela converge lentamente para o valor real de Π, exigindo um número elevado de iterações para obter uma precisão significativa.

Este trabalho tem como objetivo implementar um programa em C que calcule uma aproximação de Π usando a série de Leibniz, variando o número de iterações e medindo o tempo de execução. Além disso, será feito um estudo sobre como a precisão da aproximação melhora com o aumento do número de iterações. O código completo pode ser visto no repositório disposto na plataforma GitHub.

2 Metodologia

Para realizar o cálculo de Π, utilizamos a série de Leibniz, dada por:

$$\pi = 4 \sum_{i=0}^{\infty} \frac{(-1)^i}{2i+1}$$

Essa série converge lentamente para o valor de Π, o que implica que, para valores maiores de precisão, muitas iterações são necessárias. A fórmula geral para o cálculo de cada termo da soma é:

$$\text{termo}_i = \frac{4}{2i+1}$$

onde o sinal alterna entre positivo e negativo, dependendo do índice i. O código implementado

em C realiza o cálculo de Π para diferentes quantidades de iterações, e o tempo de execução é medido utilizando a função $gettimeofday()$ da biblioteca $< sys/time.h >$.

Foram realizadas medições para cinco diferentes quantidades de iterações. O erro absoluto foi calculado como a diferença entre o valor real de Π, Π_{real} = 3.14159265358979323846, e o valor aproximado obtido.

2.1 Implementação

Assim como no projeto passado, para medir o tempo de execução, foi recomendado em sala de aula o uso da função $gettimeofday()$, que captura o tempo total de execução $(wall - clocktime)$, incluindo pausas, esperas e operações de I/O. Essa abordagem difere da função $clock()$, que mede apenas o tempo de CPU efetivamente utilizado pelo processo, ignorando períodos de inatividade, como espera por I/O ou suspensão do programa. Essa função é chamada no inicio do for e no final e para o cálculo fina do tempo de execução realizamos uma subtração do tempo final pelo inicial.

Antes de iniciar os cálculos definimos uma função que realiza a serie de Leibniz, sendo ela:

```c
#define PI_REAL 3.14159265358979323846  // Valor real de

// Funcao para calcular Pi usando a serie de Leibniz
double calcular_pi(int iteracoes) {
    double pi = 0.0;
    for (int i = 0; i < iteracoes; i++) {
        double termo = (double) 4 / (2 * i + 1);
        if (i % 2 == 0) {
            pi += termo;
        } else {
            pi -= termo;
        }
    }
    return pi;
}
```

A Função recebe como argumento o número de iterações a ser utilizado para calcular a aproximação de Π usando a série de Leibniz. O cálculo do termo da série é feito dentro do laço for, onde Para cada iteração i, o termo é dado por

$$\text{termo}_i = \frac{4}{2i + 1}$$

Se i for par ($i\%2 == 0$), o termo é somado a Π, caso contrário, o termo é subtraído.

Iniciamos a função *main*, definindo o conjunto de quantidade de interações para a aproximação do valor de Π por meio de um array $iteracoes[]$ e logo em seguida, definimos e $qtd\_iteracoes$ como uma variável que irá calcular o número de elementos no array iterações.

```c
int main() {
    int iteracoes[] = {10, 100, 1000, 10000, 1000000};  // Diferentes quantidades de iteracoes
    int qtd_iteracoes = sizeof(iteracoes) / sizeof(iteracoes[0]);

    printf("Iteracoes\tPi Aproximado\t\tErro Absoluto\t\tTempo (ms)\n");
    printf("-----------------------------------------\n");

    for (int i = 0; i < qtd_iteracoes; i++) {
        int n = iteracoes[i];

        // Medir tempo inicial
        struct timeval inicio, fim;
        gettimeofday(&inicio, NULL);

        // Calcular aproximacao de Pi
        double pi_aprox = calcular_pi(n);

        // Medir tempo final
        gettimeofday(&fim, NULL);

        // Calcular tempo decorrido em milissegundos
        double tempo = (fim.tv_sec - inicio.tv_sec) * 1000.0 + (fim.tv_usec - inicio.tv_usec) / 1000.0;

        // Calcular erro absoluto
        double erro = fabs(PI_REAL - pi_aprox);

        printf("%d\t\t%.10f\t%.10f\t%.4f ms\n", n, pi_aprox, erro, tempo);
    }

    return 0;
}
```

O laço for percorre o array iterações, calculando a aproximação de Π para cada número de iterações. gettimeofday é usada para capturar o tempo de início (inicio) e o tempo final (fim) da execução da função calcular_pi.

O tempo de execução é calculado em milissegundos. A variável erro = fabs(PI_REAL - pi_aprox), calcula o erro absoluto entre o valor real de Π (PI_REAL) e o valor aproximado (pi_aprox).

3 Resultados e Discussão

A tabela a seguir apresenta o tempo de execução após a implementação do programa para cada número de iterações:

Número de Interações	Tempo (ms)
10000	0.0000000000
100000	0.9990000000
1000000	4.2240000000
10000000	57.6180000000
100000000	2742.4770000000

Fica nítido que, com o aumento do número de interações, o tempo de execução também apresenta um crescimento constante. Além do tempo, à medida que o número de interações aumenta, a quantidade de dígitos semelhantes entre o Π aproximado e o Π real também cresce de forma linear. Podemos observar esse crescimento na Figura 1.

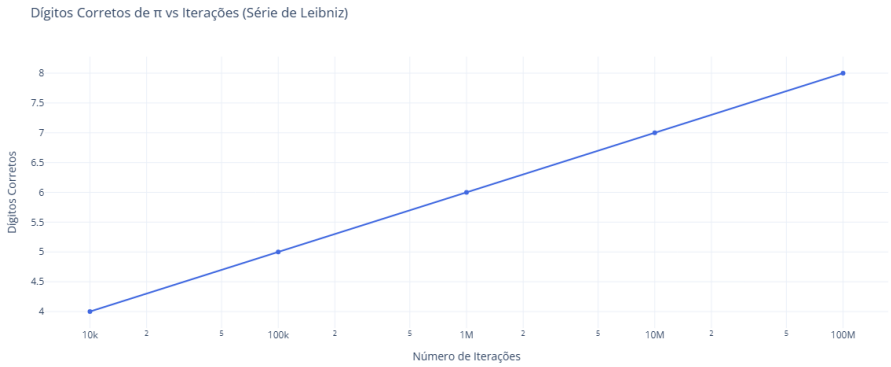

Figura 1: Gráfico de desempenho do tempo de execução para os dois casos estudados.

A precisão do valor de π melhora à medida que o número de iterações aumenta. Com 10.000 iterações, apesar do tempo de execução ser baixo, a quantidade de dígitos corretos é mínima. Por outro lado, com 1.000.000 de iterações, embora o tempo de execução atinja 2742,47 milissegundos, o número de dígitos corretos ultrapassa cinco.

4 Conclusão

Os resultados obtidos confirmam o comportamento esperado para a série de Leibniz: a precisão aumenta com o número de iterações, mas o tempo de execução também cresce significativamente. Esse fenômeno é uma característica de muitas abordagens numéricas que buscam maior precisão, o que exige mais recursos computacionais.

A implementação de algoritmos paralelizados, como discutido por Pacheco (2011), pode ser uma solução para mitigar o aumento exponencial do tempo de execução à medida que as iterações crescem, especialmente em contextos onde grandes volumes de cálculos precisam ser realizados de maneira eficiente.