Programação Paralela
Tarefa 3 - Aproximação matemática de π

Anelma Silva da Costa
UFRN - DCA
anelma.costa.100@ufrn.edu.br

I - Introdução

Implemente um programa em C que calcule uma aproximação de π usando uma série matemática, variando o número de iterações e medindo o tempo de execução. Compare os valores obtidos com o valor real de π e analise como a acurácia melhora com mais processamento. Reflita sobre como esse comportamento se repete em aplicações reais que demandam resultados cada vez mais precisos, como simulações físicas e inteligência artificial.

O número π (pi) é uma constante matemática fundamental, definida como a razão entre a circunferência de um círculo e seu diâmetro. Sua importância transcende a geometria, permeando diversas áreas da ciência e da engenharia, como física computacional, processamento de sinais e inteligência artificial. Embora π seja um número irracional, diversas técnicas numéricas permitem aproximá-lo com precisão crescente, como o uso de séries matemáticas convergentes.

Neste relatório, apresenta-se a implementação de um programa em C que calcula uma aproximação de π por meio de uma série matemática, investigando a relação entre o número de iterações, o tempo de execução e a acurácia do resultado obtido. A análise comparativa com o valor real de π permite avaliar como o aumento do processamento influencia a precisão da estimativa. Além disso, discute-se como esse comportamento se assemelha a aplicações práticas que demandam maior poder computacional para alcançar resultados mais precisos, como simulações de sistemas físicos complexos e algoritmos de aprendizado de máquina.

A reflexão sobre esse paralelo evidencia um princípio central na computação científica: a busca por maior exatidão frequentemente implica custos adicionais de tempo e recursos, exigindo um equilíbrio entre desempenho e precisão conforme as necessidades do problema em questão. Assim, este estudo não apenas ilustra um método numérico clássico, mas também oferece insights sobre otimização computacional em contextos reais.

II - Teoria

Em computação científica, acurácia refere-se à proximidade de um resultado calculado em relação ao valor real, enquanto precisão está relacionada à reprodutibilidade das medições. O processamento (tempo de CPU, número de operações) atua como um recurso que, quando aumentado, permite:

- Mais iterações em métodos iterativos (ex: séries numéricas).
- Amostragem mais densa em métodos probabilísticos (ex: Monte Carlo).
- Malhas mais refinadas em métodos discretizados (ex: Elementos Finitos).

A melhoria da acurácia com maior processamento é uma consequência direta da lei dos grandes números (em métodos estatísticos) e da convergência assintótica (em métodos numéricos).

Muitos algoritmos numéricos que aproximam soluções para problemas matemáticos, como o cálculo de π via séries de Leibniz ou outras séries de convergência lenta, dependem de uma sequência de aproximações sucessivas que se tornam mais precisas à medida que o número de iterações aumenta. A convergência de uma série é o processo em que a soma dos termos se aproxima de um valor específico à medida que o número de termos (ou iterações) aumenta.

No caso específico do cálculo de π pela série de Leibniz (A série utilizada nessa tarefa):

$\boldsymbol{\pi} \approx 4\,(1\ -\ 1/3\ +\ 1/5\ -\ 1/7\ +\ \ldots)$

a cada termo adicional, o valor de π se aproxima mais do valor real, mas a convergência da série é lenta, ou seja, a cada iteração a diferença entre o valor aproximado e o valor real diminui de forma mais gradual. Esse fenômeno é comum em várias séries numéricas, como a série de Taylor, onde a precisão melhora conforme aumentamos o número de termos, mas com um ganho progressivamente menor.

Portanto, quanto mais iterações (ou passos) o algoritmo realiza, mais próximo ele chega do valor exato da solução, melhorando a acurácia.

Embora o aumento do número de iterações melhore a precisão, ele também aumenta o custo computacional. Isso ocorre porque, em muitos casos, a melhoria na precisão exige mais operações aritméticas, o que aumenta o tempo de processamento necessário. Além disso, a complexidade computacional de um algoritmo pode crescer com o número de iterações, especialmente se o algoritmo for não linear ou envolver cálculos pesados.

Em problemas de simulação e modelagem numérica, como no cálculo de π pela série de Leibniz, o custo do aumento da precisão é proporcional ao número de termos calculados. Esse aumento é muitas vezes linear ou quadrático, o que significa que, para obter uma precisão significativamente melhor, o tempo de execução do programa aumentará proporcionalmente.

Há uma relação direta entre o tempo de processamento e a precisão dos resultados obtidos por métodos numéricos. Isso pode ser ilustrado de duas maneiras principais:

- Melhoria incremental da precisão: À medida que o número de iterações ou a resolução aumenta, a melhoria na precisão é incremental, ou seja, a diferença entre o valor aproximado e o valor exato diminui em uma taxa mais lenta. No entanto, o custo computacional aumenta, pois o número de operações necessárias cresce.

- Limitações práticas: Em aplicações computacionais reais, como simulações físicas e algoritmos de inteligência artificial, a melhoria da acurácia nem sempre justifica o aumento exponencial no tempo de processamento. Existe um ponto de diminuição dos retornos, onde os ganhos em precisão se tornam desprezíveis em relação ao aumento no tempo de execução. Isso é conhecido como trade-off entre precisão e eficiência computacional.

Esse trade-off é fundamental para a eficiência computacional em sistemas reais, onde a precisão necessária deve ser balanceada com os recursos computacionais disponíveis. Em algumas aplicações, um valor mais preciso de π, por exemplo, pode ser desnecessário após um número razoável de iterações, o que implica que o benefício de mais iterações não compensa o custo computacional.

Casos de Uso em Aplicações Reais

- **Simulações Físicas:**

Dinâmica Molecular: Cálculos de forças interatômicas requerem passos de tempo infinitesimais (10^-15s) para evitar erros de integração.

Clima e Turbulência: Modelos como LES (Large Eddy Simulation) demandam malhas com bilhões de pontos para capturar vórtices críticos.

- **Inteligência Artificial:**

Treinamento de Redes Neurais: Acurácia em visão computacional (ex: ImageNet) melhorou com:

- Mais dados (Lei de Power Scaling): Dataset de 1M → 100M imagens.

- Mais parâmetros: Modelos como GPT-3 (175 bilhões de parâmetros) exigem milhares de GPUs.

- **Medicina de Precisão:**

Genômica: Alinhamento de sequências de DNA com algoritmos como BLAST requer heurísticas para balancear acurácia e tempo.

III - Aplicação Prática

Para visualização dos conceitos apresentados até aqui, foi desenvolvido um código em C no qual é calculada a aproximação de $\boldsymbol{\pi}$ através da série de Leibniz variando o número de interações.

Com o objetivo de observar a comportamento da aproximação e tempo de execução com a variação das interações, foi implementado no mesmo funções que a cada interação verifica quantos dígitos da aproximação é igual ao número real de π e o tempo de execução da quantidade de interações atuais.

A imagem abaixo exibe o vetor criado para alterar o número de interações.

```
int iteracoes[] = {10, 50, 100, 200, 400, 800, 1000, 2000, 4000, 8000,
                   10000, 20000, 40000, 60000, 80000, 100000, 200000,
                   400000, 800000, 1000000, INTERACOES};
```

Fez-se uso da biblioteca <math.h> da linguagem C para se obter o valor de π, isso ocorreu através do uso da M_PI, assim não foi necessário definir o valor no código.

A figura abaixo exibe o laço for responsável pelo objetivo principal da tarefa. Nele é onde está alocada as chamadas das funções responsáveis pelo cálculo da aproximação de π e a comparação da quantidade de dígitos iguais dos dois valores.

```
for (int i = 0; i < num_testes; i++) {
    struct timespec start, end;
    clock_gettime(CLOCK_MONOTONIC, &start);

    double pi_aproximado = calcular_pi(iteracoes[i]);

    clock_gettime(CLOCK_MONOTONIC, &end);

    // Calcula tempo em segundos com precisão de nanossegundos
    double tempo_execucao = (end.tv_sec - start.tv_sec) +
                            (end.tv_nsec - start.tv_nsec) / 1e9;

    int digitos_corretos = contar_digitos_corretos(pi_aproximado, M_PI);

    printf("Iteracoes: %9d | Aproximacao: %.15f\n", iteracoes[i], pi_aproximado);
    printf("Digitos corretos: %d\n", digitos_corretos);
    printf("Tempo: %.9f segundos\n\n", tempo_execucao);
}
```

IV - Teoria x Resultados Práticos

Como já mencionado anteriormente neste documento, de forma teórica, com o aumento das interações pressupõe-se que a acurácia melhore, mas isso custará maior processamento.

Ao analisar o resultado obtido com a compilação do código, de fato isso se propagou para a realidade. Ao chegar ao número máximo de interações que foi definido, neste caso, 100000000, temos uma acurácia de 7 casas decimais iguais ao π, mas um processamento bem mais demorado se comparado aos outros números.

A série de Leibniz converge como esperado, mas de forma lenta, como prediz a teoria. A cada termo adicionado, a aproximação melhora, mas são necessários muitos termos para obter precisão significativa.

Em relação ao tempo como cada iteração realiza operações simples (divisão, adição, alternância de sinal), o tempo deve crescer linearmente com o número de iterações.

A série de Leibniz não é eficiente para cálculos de alta precisão, mas serve como um bom exemplo didático de convergência lenta.

A imagem abaixo evidencia um trecho final do resultado obtido com o código.

```
Iteracoes:    400000 | Aproximacao: 3.141590153589744
Digitos corretos: 6
Tempo: 0.001574900 segundos

Iteracoes:    800000 | Aproximacao: 3.141591403589717
Digitos corretos: 6
Tempo: 0.003251000 segundos

Iteracoes:   1000000 | Aproximacao: 3.141591653589774
Digitos corretos: 6
Tempo: 0.004061600 segundos

Iteracoes: 100000000 | Aproximacao: 3.141592643589326
Digitos corretos: 8
Tempo: 0.395850400 segundos
```

V - Conclusões

Este experimento não apenas validou as propriedades matemáticas da série de Leibniz, mas também serviu como um modelo simplificado para entender desafios em projetos computacionais avançados. A análise dos dados destacou a importância de:

- Selecionar algoritmos adequados ao problema.
- Monitorar trade-offs entre precisão e eficiência.
- Explorar técnicas de aceleração quando a precisão é crítica.

Para futuros trabalhos, recomenda-se estender a análise para outros métodos de cálculo de π (ex: Monte Carlo) ou investigar otimizações baseadas em hardware.

Em resumo, a atividade proporcionou uma compreensão tangível de como a teoria numérica se manifesta na prática computacional, preparando o terreno para abordagens mais sofisticadas em problemas científicos e de engenharia.

VI - Referências

- Livro: An introduction to parallel programming .
- Conteúdo ministrado em sala de aula.
- IA's (DeepSeek e ChatGPT)