Custo Computacional vs. Precisão do Resultado

O trade-off entre o custo computacional e a precisão do resultado

Werbert Arles de Souza Barradas

Universidade Federal do Rio Grande do Norte (UFRN)
Disciplina de Programação Paralela - DCA3703

20 de agosto de 2025

Introdução

O Trade-Off Fundamental

O objetivo é demonstrar empiricamente um dos princípios mais importantes da computação de alto desempenho: a relação direta entre **custo computacional (tempo) e a precisão do resultado**.

O Experimento Modelo

Este estudo utiliza o cálculo de π através de uma série matemática para visualizar este trade-off. Ao variar o número de iterações, mede-se como o tempo de processamento aumenta em troca de uma diminuição no erro da aproximação.

Relevância

Esta análise serve como um modelo simplificado para entender desafios em domínios como simulações, renderização gráfica e IA.

Metodologia: O Cálculo de π

A Série de Leibniz

Foi utilizada a série de Leibniz, escolhida pela sua convergência lenta, que torna a relação entre iterações e precisão mais evidente.

$$\frac{\pi}{4} = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = 1 - \frac{1}{3} + \frac{1}{5} - \cdots$$

Foco do Teste

O código foi compilado sem otimizações (`-O0`) para isolar o ganho de desempenho obtido **apenas pela reestruturação do código** e pela melhor exploração do hardware (ILP).

Metodologia: Duas Implementações em C

Versão Sequencial

Cria uma dependência de dados no acumulador `sum`, limitando o ILP.

```
double calculate_pi(long long num_iter) {
    double sum = 0.0;
    int sign = 1;
    for (long long i = 0; i < num_iter; i++) {
        double term = (double)sign / (2.0*i+1.0);
        sum += term;
        sign *= -1;
    }
    return 4.0 * sum;
}
```

Metodologia: Duas Implementações em C

Versão Otimizada (ILP)

Usa 4 acumuladores para quebrar a dependência, permitindo que a CPU execute 4 operações em paralelo.

```
double calculate_pi_ilp(long long num_iter) {
    double s1=0.0, s2=0.0, s3=0.0, s4=0.0;
    for (long long i=0; i<num_iter; i+=4) {
        s1 += 1.0 / (2.0*(i+0)+1.0);
        s2 -= 1.0 / (2.0*(i+1)+1.0);
        s3 += 1.0 / (2.0*(i+2)+1.0);
        s4 -= 1.0 / (2.0*(i+3)+1.0);
    }
    return 4.0 * (s1+s2+s3+s4);
}
```

Speedup da Versão ILP vs. Sequencial (Gráfico de Linha)

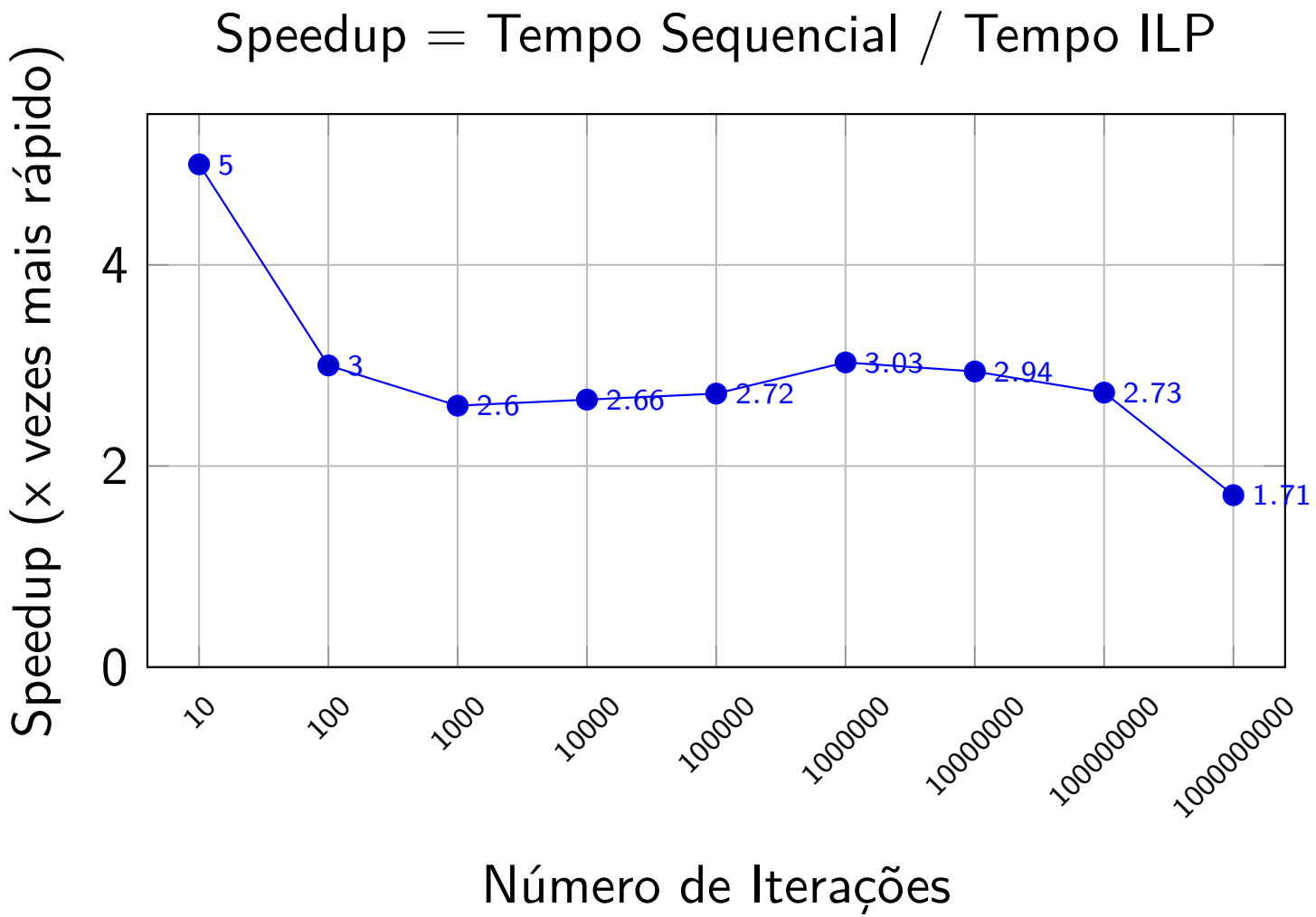

Figure: A versão otimizada para ILP mostra um ganho de desempenho consistente.

Relação entre Tempo de Execução e Erro na Aproximação de π

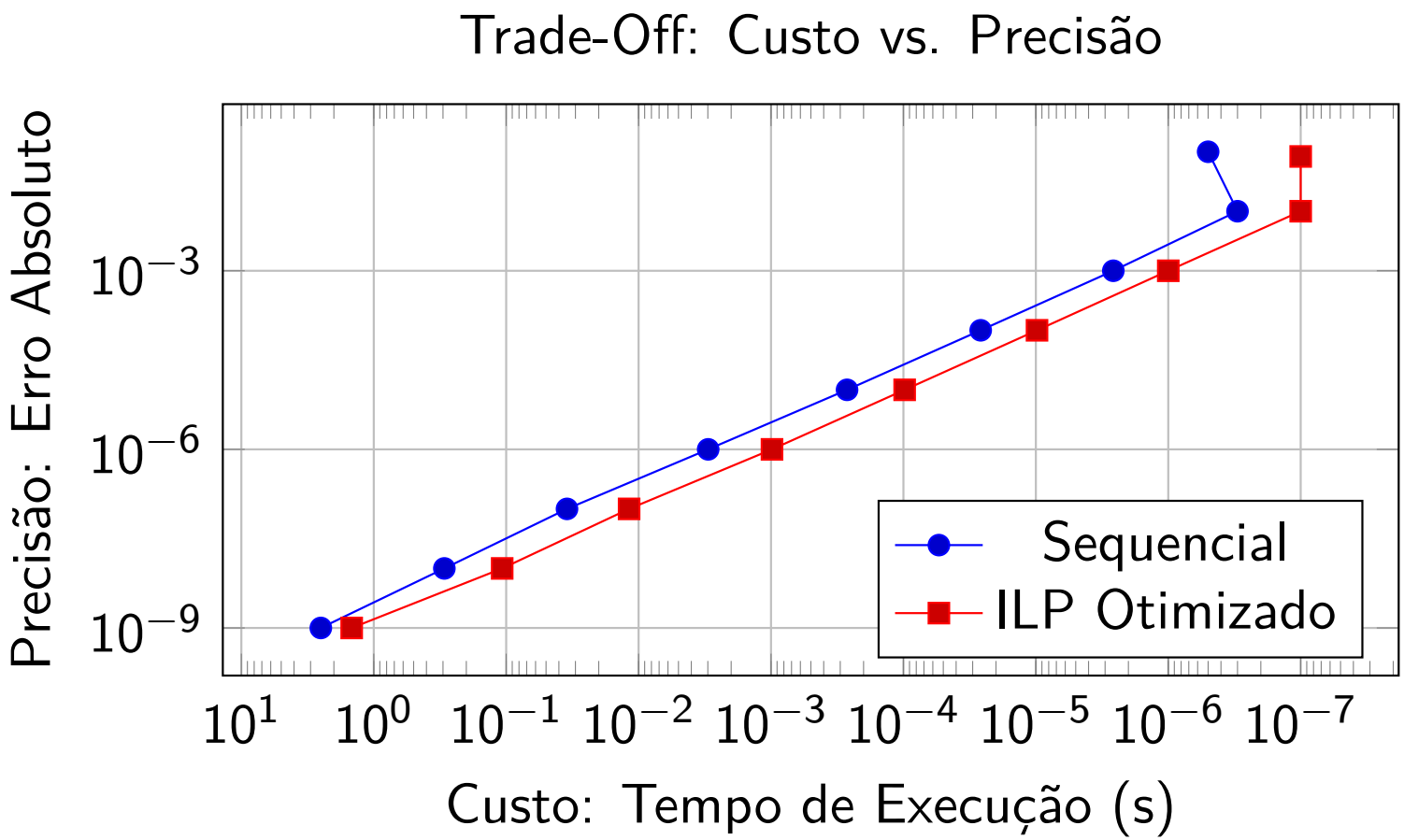

Figure: Para alcançar o mesmo nível de precisão (erro), a versão ILP exige um custo (tempo) muito menor.

Uma Analogia Prática e Visual

Objetivo

Demonstrar o mesmo trade-off entre custo (tempo) e precisão (qualidade) do cálculo de π, mas num domínio prático: processamento de imagens.

Figure: Imagem original em alta resolução (3400x2216) usada no teste.

O Experimento:

- Redimensionar a imagem para um tamanho menor (800x521).
- Utilizar quatro algoritmos de interpolação com diferentes complexidades:
  1. **Nearest Neighbor** (Rápido, baixa qualidade).
  2. **Bilinear** (Intermediário).
  3. **Bicubic** (Lento, alta qualidade).
  4. **Lanczos** (Muito lento, máxima qualidade).

Comparando a Qualidade dos Resultados

Figure: Zoom numa área de detalhe da imagem redimensionada por cada algoritmo.

Comparando a Qualidade dos Resultados

- **Nearest Neighbor:** Produz um forte serrilhado (*aliasing*).
- **Bilinear:** Suaviza o serrilhado, mas causa um borrão (*blur*), perdendo nitidez.
- **Bicubic e Lanczos:** Oferecem um resultado superior, preservando a nitidez das bordas. O Lanczos mantém os detalhes mais finos.

Medindo o Tempo para Atingir a Qualidade

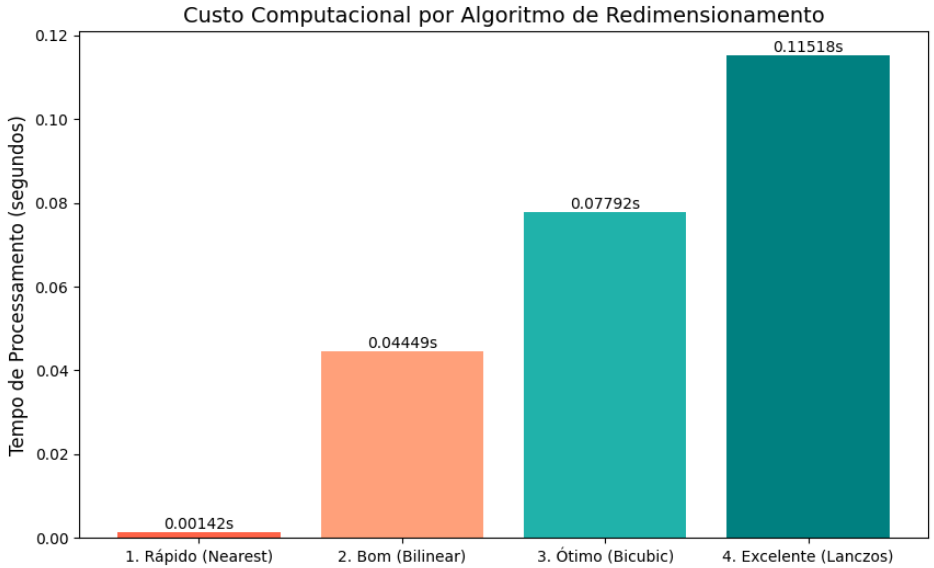

Figure: Tempo de execução de cada algoritmo de redimensionamento.

Correlação Custo-Qualidade

Os dados mostram uma correlação direta: para obter a precisão visual superior do algoritmo Lanczos, é necessário um investimento computacional drasticamente maior.

Conclusão do Estudo de Caso

- Assim como no cálculo de π, onde mais iterações (custo) levam a um erro menor (precisão), no processamento de imagens, algoritmos mais complexos (custo) resultam em maior fidelidade visual (precisão).
- A escolha do método ideal depende sempre dos requisitos da aplicação.

Conclusão Geral do Projeto

Tese Comprovada

Este projeto demonstrou com sucesso, tanto de forma quantitativa (cálculo de π) quanto qualitativa (processamento de imagens), o trade-off fundamental entre custo computacional e precisão do resultado.

- **No cálculo de π:** Foi provado que a reestruturação do código para explorar o ILP (paralelismo em nível de instrução) permite alcançar a mesma precisão em um tempo significativamente menor, otimizando o uso do hardware.
- **No processamento de imagens:** Foi demonstrado que diferentes algoritmos representam pontos distintos na curva custo-benefício, onde maior fidelidade visual exige um custo de processamento exponencialmente maior.

pi_accuracy_graficos.c

```c

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h> // Necessário para M_PI

double calculate_pi_sequential(long long num_iterations) {
    double sum = 0.0;
    int sign = 1;

    for (long long i = 0; i < num_iterations; i++) {
        double term = (double)sign / (2.0 * i + 1.0);
        sum += term;
        sign *= -1;
    }

    return 4.0 * sum;
}


double calculate_pi_ilp(long long num_iterations) {
    double sum1 = 0.0, sum2 = 0.0, sum3 = 0.0, sum4 = 0.0;

    for (long long i = 0; i < num_iterations; i += 4) {
        sum1 += 1.0 / (2.0 * i + 1.0);
        sum2 -= 1.0 / (2.0 * i + 3.0);
        sum3 += 1.0 / (2.0 * i + 5.0);
        sum4 -= 1.0 / (2.0 * i + 7.0);
    }

    double total_sum = sum1 + sum2 + sum3 + sum4;

    return 4.0 * total_sum;
}


int main() {
    long long iterations[] = {
        10, 100, 1000, 10000, 100000, 1000000,
        10000000, 100000000, 1000000000
    };
    int num_tests = sizeof(iterations) / sizeof(iterations[0]);

    // Abre o ficheiro CSV para escrita
    FILE *csv_file = fopen("pi_results.csv", "w");
    if (csv_file == NULL) {
        perror("Erro ao abrir o ficheiro CSV");
        return 1;
    }

    // Escreve o cabeçalho do CSV
```

```c
    fprintf(csv_file, "Metodo,Iteracoes,PiCalculado,ErroAbsoluto,Tempo_s\n");
    printf("A gerar resultados em pi_results.csv...\n");

    // --- Testes para a Versão Sequencial ---
    for (int i = 0; i < num_tests; i++) {
        long long current_iterations = iterations[i];
        struct timespec start_time, end_time;

        clock_gettime(CLOCK_MONOTONIC, &start_time);
        double pi_approximation = calculate_pi_sequential(current_iterations);
        clock_gettime(CLOCK_MONOTONIC, &end_time);

        double time_spent = (end_time.tv_sec - start_time.tv_sec) +
                            (end_time.tv_nsec - start_time.tv_nsec) / 1e9;
        double error = fabs(M_PI - pi_approximation);

        // Escreve os dados no ficheiro CSV
        fprintf(csv_file, "Sequencial,%lld,%.18f,%.18f,%.9f\n",
               current_iterations, pi_approximation, error, time_spent);
    }

    // --- Testes para a Versão Otimizada com ILP ---
    for (int i = 0; i < num_tests; i++) {
        long long current_iterations = iterations[i];
        struct timespec start_time, end_time;

        clock_gettime(CLOCK_MONOTONIC, &start_time);
        double pi_approximation = calculate_pi_ilp(current_iterations);
        clock_gettime(CLOCK_MONOTONIC, &end_time);

        double time_spent = (end_time.tv_sec - start_time.tv_sec) +
                            (end_time.tv_nsec - start_time.tv_nsec) / 1e9;
        double error = fabs(M_PI - pi_approximation);

        // Escreve os dados no ficheiro CSV
        fprintf(csv_file, "ILP_Otimizado,%lld,%.18f,%.18f,%.9f\n",
               current_iterations, pi_approximation, error, time_spent);
    }

    // Fecha o ficheiro
    fclose(csv_file);

    printf("Resultados guardados com sucesso em pi_results.csv\n");
    printf("Valor de referência de M_PI: %.18f\n", M_PI);

    return 0;
}

```