Relatório Trabalho A2

Aluno: João Pedro Silva de Lima

Objetivo

A ideia deste trabalho é utilizar a biblioteca MPI para realizar um processamento paralelo entre diferentes processadores, estes podendo ser núcleos em uma única CPU ou até mesmo um *cluster* com vários processadores simultâneos.

Utilizando a força combinada de vários núcleos, espera-se que o tempo de processamento total de uma aplicação seja reduzido exponencialmente, aproveitando ao máximo a capacidade de processar informações de uma CPU que possua mais de um núcleo.

Neste trabalho, criei um código que calcula a área de uma função utilizando a sua integral, recebendo como parâmetros a precisão e os valores de x inicial e final, necessários para que este cálculo seja realizado. Também é possível selecionar o método de cálculo da área, utilizando retângulos ou trapezoides, para uma maior/menor precisão nos resultados.

Código

#include <stdio.h>

#include <mpi.h>

double function(double x) {

    return x \* x;  // Função: f(x) = x^2

}

double integral(double a, double b, int n, int rank, int size) {

    double h = (b - a) / n;  // Largura do espaço total dividido pelo n de secções

    double local\_sum = 0.0;

    // Controle da quantidade de intervalos para cada nó processar

    int local\_n = n / size;

    int local\_start = rank \* local\_n;

    int local\_end = local\_start + local\_n;

    // Ajuste para tratar intervalos que "sobram" na divisão inteira

    if (rank == size - 1) {

        local\_end = n;

        local\_n = local\_end - local\_start;

    }

    for (int i = local\_start; i < local\_end; i++) {

        double x1 = a + i \* h;

        double x2 = a + (i + 1) \* h;

        double y1 = function(x1);

        double y2 = function(x2);

        double x = (x1 + x2)/2;

        double y = function(x);

        // Utilizando trapezóides

        local\_sum += (y1 + y2) \* h / 2.0;

        // Utilizando retângulos

        // local\_sum += y \* h;

    }

    double global\_sum = 0.0;

    MPI\_Reduce(&local\_sum, &global\_sum, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

    return global\_sum;

}

int main(int argc, char\*\* argv) {

    MPI\_Init(&argc, &argv);

    int size, rank; // Número de processos e o rank do processo atual

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &size);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

    double a = 0.0;  // Mínimo no eixo x

    double b = 200.0;  // Máximo no eito x

    int n = 10000;     // Quantidade de intervalos

    double start, finish;

    start = MPI\_Wtime();

    double result = integral(a, b, n, rank, size);

    finish = MPI\_Wtime();

    // Quando rank==0 quer dizer que o processamento já ofi realizado em todos os nós,

    // chegando então no nó "master"

    if (rank == 0) {

        printf("Área entre os pontos %lf e %lf: %f\n", a, b, result);

        printf("Tempo de execução: %lfs\n\n", finish-start);

    }

    MPI\_Finalize();

    return 0;

}

Output



Resultados

Speedup:

|  |  |  |  |
| --- | --- | --- | --- |
| Formato da secção | Nº de nós | Precisão (nº de secções) | Tempo (s) |
| Trapezoide | 1 | 100.000 | 0.000611 |
| Trapezoide | 1 | 9.999.999 | 0.613471 |
| Trapezoide | 4 | 100.000 | 0.001461 |
| Trapezoide | 4 | 9.999.999 | 0.064288 |

Análise

Utilizando uma função que calcula a potência 2 de uma variável eu encontro o gráfico no plano cartesiano que a define. Para calcular a área abaixo do gráfico dentro de dois pontos conhecidos (chamado também de Integral Definida), Devemos transformar a área do gráfico em várias formas geométricas (trapézios ou retângulos) com larguras cada vez menores, tendendo ao infinito. Quando maior a quantidade destas formas geométricas, e consequentemente menor as suas larguras, mais precisa será a área do gráfico.

Neste código criamos a função que calcula o valor de y para um dado x, chamada de *function*. Quando vamos realizar o cálculo da integral, de acordo com a precisão definida pelo usuário, selecionamos os valores de x e aplicamos *function* nele. De acordo com a forma geométrica escolhida, ele será então calculado e adicionado a variável *local\_sum*.

Para que esse código seja *paralelizável,* importamos a biblioteca <MPI.h> e atribuímos, a partir do *MPI\_COMM\_WORLD* o total de nós que temos disponíveis e o rank do nó que será atribuído ao rodar o código. Então, é definido um valor de iterações para cada nó, valor este resultante da divisão entre o valor da precisão e o total de nós. Após isso, calculamos então o y de cada um dos pontos, dentro da divisão em cada nó, somando-o e utilizando a função *MPI\_Reduce* para que o resultado dos nós seja atribuído a uma variável única, através do modo *MPI\_SUM*, que soma os valores enviados na primeira posição da API para o endereço de memória na segunda posição.