Projeto PCD: Simulação e Análise de Modelos de Difusão de Contaminantes em Água

Profs. Álvaro e Denise (Turmas I e N)

Objetivo: Criar uma simulação que modele a difusão de contaminantes em um corpo d'água (como um lago ou rio), aplicando conceitos de paralelismo para acelerar o cálculo e observar o comportamento de poluentes ao longo do tempo. O projeto investigará o impacto de OpenMP, CUDA e MPI no tempo de execução e na precisão do modelo.

Etapas do Projeto

1. Estudo do Modelo de Difusão

- Estudar a Equação de Difusão/Transporte transiente, representada por:

$$\frac{\partial C}{\partial t} = D \cdot \nabla^2 C$$

onde:

- C é a concentração do contaminante,
- t é o tempo,
- D é o coeficiente de difusão,
- $\nabla^2 C$ representa a taxa de variação da concentração no espaço.

- A equação diferencial pode ser aproximada (discretizada) no tempo e no espaço usando diferenças finitas em uma grade bidimensional, onde a discretização no espaço implica que cada célula da grade atualiza seu valor com base nas células vizinhas em cada iteração. O cálculo da grade atualizada deverá se repetir para que sejam feitas várias interações discretas no tempo.

2. Configuração do Ambiente e Parâmetros da Simulação

- Configurar uma grade 2D onde cada célula representa a concentração de contaminantes em uma região do corpo d'água.
- Definir o coeficiente de difusão (D), as condições de contorno (por exemplo, bordas onde o contaminante não se espalha) e as condições iniciais (como uma área de alta concentração de contaminante).
- Definir uma quantidade fixa de interações no tempo como 1000 iterações.

3. Implementação com OpenMP (Simulação Local em CPU)

- Usar OpenMP para paralelizar o cálculo de difusão entre os núcleos da CPU. Cada núcleo processa uma parte da grade, aplicando as regras de difusão às células sob sua responsabilidade.
- **Entrega 1**: demonstrar o código em OpenMP e apresentar avaliação de desempenho com relação à versão sequencial.

4. Implementação com CUDA (Simulação em GPU)

- Implementar a simulação em CUDA, onde cada célula da grade é processada por uma thread independente na GPU, utilizando um esquema de diferenças finitas para calcular o laplaciano de (C).
- A execução em GPU permite simular uma grade maior e observar o ganho de desempenho com CUDA.
- **Entrega 2**: demonstrar o código em CUDA e apresentar avaliação de desempenho com relação às versões anteriores.

5. Distribuição com MPI (Simulação em Larga Escala)

- Dividir a grade em sub-regiões e distribuir o processamento entre várias máquinas usando MPI.
- Cada máquina processa uma seção do corpo d'água e troca informações nas bordas com as máquinas vizinhas para garantir a continuidade da difusão de contaminantes entre as regiões.
- **Entrega 3**: demonstrar o código em MPI hibrido (pode incluir trechos em OpenMP e CUDA) e apresentar avaliação de desempenho com relação às versões anteriores, porém destacando a escalabilidade possível apenas com MPI.

6. Artigo científico e Discussão dos Resultados

- Criar gráficos que mostrem a evolução da concentração ao longo do tempo e comparar o tempo de execução entre as implementações.
- Discutir as vantagens e limitações de cada abordagem, observando a escalabilidade, precisão e aplicabilidade em simulações ambientais.

- Demonstrar visualmente os resultados que comprovem a corretude da simulação.
- **Entrega Final**: entregar o resultado final no formato de artigo científico (modelo a ser disponibilizado).

Ponto de Partida para a Implementação da Equação

Para aproximar a Equação de Difusão, podemos usar a seguinte fórmula de diferenças finitas central:

$$C_{i,j}^{t+1} = C_{i,j}^{t} + D \cdot \Delta t \left(\frac{C_{i+1,j}^{t} + C_{i-1,j}^{t} + C_{i,j+1}^{t} + C_{i,j-1}^{t} - 4 \cdot C_{i,j}^{t}}{\Delta x^2} \right)$$

Abaixo está um trecho de código em C para uma implementação sequencial simples, o qual calcula a difusão do contaminante em uma grade de 2000x2000 ao longo de 500 ciclos. A concentração inicial está configurada no centro da grade, e o coeficiente de difusão (D) pode ser ajustado conforme necessário.

Características do Projeto

- **Caráter Científico:** Esse projeto aplica uma abordagem científica a um problema ambiental relevante.
- **Exploração e Análise de Desempenho:** Permite uma análise detalhada das implementações em OpenMP, CUDA e MPI, com foco em desempenho e escalabilidade.
- **Estrutura de Redação Científica:** O projeto estimula a criação de um artigo científico, semelhante a um trabalho de iniciação científica.

O grupo tem a liberdade de alterar o código, bem como alterar parâmetros/opções de compilação, para melhorar o seu desempenho, mantendo o seu objetivo, ou seja, calcular, sucessivamente, os novos valores para a matriz C_new, conforme a discretização citada acima.

Em caso de dúvida, será requisitado aos autores para que demonstrem e expliquem o funcionamento do código no laboratório de computação.

```
%%writefile difusao.c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define N 2000  // Tamanho da grade
#define T 500 // Número de iterações no tempo
#define D 0.1  // Coeficiente de difusão
#define DELTA_T 0.01
#define DELTA_X 1.0

void diff_eq(double **C, double **C_new) { //diff_eq(double C[N][N], double C_new[N][N]) {
    for (int t = 0; t < T; t++) {
        for (int i = 1; i < N - 1; i++) {
            for (int j = 1; j < N - 1; j++) {
                C_new[i][j] = C[i][j] + D * DELTA_T * (
                    (C[i+1][j] + C[i-1][j] + C[i][j+1] + C[i][j-1] - 4 * C[i][j]) / (DELTA_X * DELTA_X)
                );
            }
        }
        // Atualizar matriz para a próxima iteração
        double difmedio = 0.;
        for (int i = 1; i < N - 1; i++) {
            for (int j = 1; j < N - 1; j++) {
                difmedio += fabs(C_new[i][j] - C[i][j]);
                C[i][j] = C_new[i][j];
            }
        }
        if ((t%100) == 0)
          printf("interacao %d - diferenca=%g\n", t, difmedio/((N-2)*(N-2)));
    }
}

int main() {

    // Concentração inicial
    double **C = (double **)malloc(N * sizeof(double *));
    if (C == NULL) {
      fprintf(stderr, "Memory allocation failed\n");
      return 1;
    }
    for (int i = 0; i < N; i++) {
      C[i] = (double *)malloc(N * sizeof(double));
      if (C[i] == NULL) {
        fprintf(stderr, "Memory allocation failed\n");
        return 1;
      }
    }
    for (int i = 0; i < N; i++) {
```

```
      for (int j = 0; j < N; j++) {
        C[i][j] = 0.;
      }
    }

    // Concentração para a próxima iteração
    double **C_new = (double **)malloc(N * sizeof(double *));
    if (C_new == NULL) {
      fprintf(stderr, "Memory allocation failed\n");
      return 1;
    }
    for (int i = 0; i < N; i++) {
      C_new[i] = (double *)malloc(N * sizeof(double));
      if (C_new[i] == NULL) {
        fprintf(stderr, "Memory allocation failed\n");
        return 1;
      }
    }
    for (int i = 0; i < N; i++) {
      for (int j = 0; j < N; j++) {
        C_new[i][j] = 0.;
      }
    }

    // Inicializar uma concentração alta no centro
    C[N/2][N/2] = 1.0;

    // Executar as iterações no tempo para a equação de difusão
    diff_eq(C, C_new);

    // Exibir resultado para verificação
    printf("Concentração final no centro: %f\n", C[N/2][N/2]);
    return 0;
}
```

```
Writing difusao.c
```

```
!rm difusao.x
!gcc difusao.c -o difusao.x
!time ./difusao.x
```

```
interacao 0 - diferenca=2.00401e-09
interacao 100 - diferenca=1.23248e-09
interacao 200 - diferenca=7.81794e-10
interacao 300 - diferenca=5.11528e-10
interacao 400 - diferenca=4.21632e-10
Concentração final no centro: 0.216512

real    0m36.053s
user    0m35.709s
sys     0m0.054s
```

```
!more /proc/cpuinfo &> processador.txt
!more processador.txt | grep model
```

```
model           : 79
model name      : Intel(R) Xeon(R) CPU @ 2.20GHz
model           : 79
model name      : Intel(R) Xeon(R) CPU @ 2.20GHz
```