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

Bruno Kenji Sato, Luiz Eduardo Casella, Nicolas Serrat

1

Resumo. *Nesse projeto criou-se uma simulação que modela a difusão de contaminantes em um corpo d'água (como um lago ou rio), aplicando conceitos de paralelismo para acelerar o cálculo e observou-se o comportamento de poluentes ao longo do tempo. O projeto investigou o impacto de OpenMP, CUDA e MPI no tempo de execução e na precisão do modelo.*

1. Características do Código

O código realizado implementa uma simulação e análise de modelos de difusão de contaminanetes em água de forma paralelizada com o uso de threads, a implementação dessas foi através da biblioteca OpenMP, que segundo [Herlihy and Shavit 2008] essa biblioteca é uma ferramenta que facilita o processo de paralelização de código em C, C++ e Fortran.

Dado o código serializado fornecido pelo professor, implementamos o uso de funções da biblioteca OpenMP, como por exemplo a função apresentada no Código 1, que define o número de threads a serem usadas nos trechos paralelizados como 4.

Código 1. Definição do número de threads

```
int main() {
    //printf("%i",omp_get_max_threads());
    omp_set_num_threads(4);
    // Inicializar as matrizes
```

Usou-se também da diretiva pragma omp parallel for: e suas variações, essa diretiva informa o compilador que o loop seguinte deve ser executado em paralelo, criando uma região paralela onde as iterações do loop são distribuidas entre as threads criadas. Como o código fornecido apresentava muitos loops for aninhados, utilizamos principalmente da diretiva pragma omp parallel for collapse(n), que tem como função paralelizar n loops aninhados permitindo que múltiplas iterações dos loops sejam executadas simultaneamente por diferentes threads como visto no Código 2.

Código 2. Paralelização de loops aninhados

```
#pragma omp parallel for collapse(2)
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            C_new[i][j] = 0.;
        }
    }
```

Além disso, usou-se também da diretiva pragma omp parallel for reduction(+:var), que vai paralelizar um loop e realizar uma operação de redução em uma variavél compartilhada entre as threads como podemos observar no Código 3, neste trecho de código a

variável difmedio será usada em uma operação de redução, ou seja, cada thread tera uma cópia local da variável difmedio e ao fim essas cópias serão somadas, em razão do +.

Código 3. Paralelização de um loop com operação de redução

```
#pragma omp parallel for reduction(+:difmedio)
        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];
            }
        }
```

2. Descrição da Avaliação de Desempenho

O código serial representa a execução sequencial de tarefas, comando por comando, sem o uso de múltiplos threads ou processos paralelos. Já o código paralelo divide essas tarefas entre diferentes threads ou núcleos, com o objetivo de reduzir o tempo total de execução. Nesse topico, aborda-se a diferença entre o tempo de execução do código da forma sequencial e da forma em paralelo, utilizando métricas como tempo total, speedup e eficiência.

Tempo total: O tempo total de execução é a duração completa de um programa, medida desde o momento em que é iniciado até a sua finalização. Ele reflete o tempo necessário para que todas as instruções sejam processadas, incluindo possíveis atrasos causados por fatores externos, como o uso compartilhado de recursos do sistema. Para medir o tempo total de execução dos códigos, utilizou-se o comando !time no Google Colab. Este comando fornece três métricas distintas:

- Tempo real: Representa o tempo decorrido entre o início e o fim da execução, incluindo pausas e interferências externas.
- Tempo de usuário: Mede o tempo efetivamente gasto pelo processador executando as instruções do programa.
- Tempo de sistema: Refere-se ao tempo gasto pelo sistema operacional gerenciando operações necessárias para a execução do programa.

Entre essas métricas, foi considerado o tempo real como base para a análise, pois ele reflete de forma mais ampla a experiência do usuário com o tempo total de execução do programa.Para obter resultados mais consistentes, foram realizadas cinco execuções de cada código, reduzindo a influência de variações pontuais. A média dos tempos reais dessas execuções foi então utilizada para representar o tempo total de execução.

Speedup: Mede o fator de melhoria ao executar o código paralelo em relação ao serial. É calculado como:

$$Speedup = \frac{TempoMédioSerial}{TempoMédioParalelo}$$

Eficiência: Avalia o aproveitamento das threads utilizadas. É calculada como:

$$Eficiência = \frac{Speedup}{NúmeroDeThreads}$$

3. Resultados

A análise dos resultados obtidos a partir da execução do código serial e das versões paralelas (utilizando 2 e 4 threads) demonstrou os seguintes tempos médios, speedup e eficiência:

Numéro de Threads	Tempo Médio (s)	Speedup	Eficiência
Código Sequencial	36,14	100%	1
Código com 2 Threads	34,85	103,7%	0,519
Código com 4 Threads	34,14	105,9%	0,265

Table 1. Tabela com Tempo Médio, Speedup e Eficiência dos códigos

O tempo médio do código serial foi 36,14 segundos, enquanto as versões paralelas obtiveram reduções para 34,85 segundos (2 threads) e 34,14 segundos (4 threads). Embora o paralelismo tenha reduzido o tempo de execução, os ganhos foram modestos.

O speedup, que mede a melhoria relativa em comparação com o código serial, apresentou valores baixos:

- 1,037 com 2 threads (3,7% mais rápido).
- 1,059 com 4 threads (5,9% mais rápido).

A eficiência, que avalia o aproveitamento das threads, foi de 51,9% para 2 threads e caiu para 26,5% para 4 threads. Essa queda reflete um aumento no custo associado à coordenação entre threads e/ou redução na escala paralelizável.

Conclusão: Embora o tempo total tenha diminuído com o aumento no número de threads, os ganhos não foram proporcionais.

References

Herlihy, M. and Shavit, N. (2008). *The Art of Multiprocessor Programming*. Elsevier, Burlington, 1st edition.