UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
IDCA3703 - PROGRAMAÇÃO PARALELA

TAREFA 11 - IMPACTO DAS CLÁUSULAS SCHEDULE E COLLAPSE
RELATÓRIO DE EXECUÇÃO

ERNANE FERREIRA ROCHA JUNIOR

NATAL/RN, 01 DE MAIO DE 2025

SUMÁRIO

1. INTRODUÇÃO

A simulação numérica de fluidos desempenha um papel fundamental em diversas áreas da engenharia e das ciências naturais, permitindo a análise de fenômenos complexos como escoamentos aerodinâmicos, dinâmica oceânica, meteorologia e processos industriais. No cerne dessa modelagem está a equação de Navier-Stokes, que descreve o movimento de fluidos viscosos a partir das leis de conservação da massa e do momento linear.

Neste trabalho, desenvolveu-se uma implementação simplificada das equações de Navier-Stokes tridimensionais, considerando exclusivamente o termo de difusão viscoso e desconsiderando a pressão e forças externas. Isso resulta em uma forma reduzida da equação, equivalente à equação de difusão vetorial da velocidade:

$$\frac{\partial \vec{u}}{\partial t} = \nu \nabla^2 \vec{u} \tag{1}$$

onde $\vec{u}$ é o campo vetorial de velocidade e ν é a viscosidade cinemática do fluido.

A solução numérica foi obtida por meio do método de diferenças finitas, utilizando uma malha cúbica uniforme em três dimensões. O objetivo principal da simulação foi observar a estabilidade do campo de velocidades e a dissipação de perturbações ao longo do tempo, características típicas de um fluido viscoso em regime sem advecção.

Além disso, visando explorar o desempenho computacional, a implementação foi paralelizada utilizando *OpenMP*, com especial atenção às cláusulas *schedule* e *collapse*, que influenciam diretamente na distribuição das tarefas entre os *threads* e na eficiência da execução paralela. Diversas combinações dessas cláusulas foram avaliadas em termos de tempo de execução, possibilitando uma análise comparativa entre as diferentes estratégias de paralelismo.

Por fim, os resultados da simulação foram salvos em arquivos binários e visualizados com a biblioteca *Plotly* em *Python*, permitindo observar graficamente a evolução temporal da perturbação no campo de velocidades.

2. METODOLOGIA

A simulação foi conduzida em uma grade tridimensional cúbica com $N = 32$ pontos em cada dimensão, representando o volume de fluido. A discretização temporal seguiu um esquema explícito com passo de tempo $\Delta t = 0,01$, enquanto o passo espacial foi definido como $\Delta x = 1.0$. O modelo utilizado é uma versão simplificada da equação de Navier-Stokes (1), restringindo-se apenas ao termo de difusão:

Para a discretização do Laplaciano no espaço, utilizou-se o esquema de cinco pontos no centro da malha tridimensional (6 vizinhos no total), aplicando diferenças finitas centradas:

$$\nabla^2 u \approx \frac{u_{i+1,j,k} + u_{i-1,j,k} + u_{i,j+1,k} + u_{i,j-1,k} + u_{i,j,k+1} + u_{i,j,k-1} - 6u_{i,j,k}}{\Delta x^2}$$

A atualização da velocidade foi feita iterativamente ao longo de 9.999 passos de tempo, com os valores armazenados em um campo auxiliar u_new e posteriormente copiados de volta para u. Duas condições iniciais foram testadas:

- **O campo homogêneo (sem perturbação)**: todo o domínio inicializado com velocidade zero;
- **Campo com perturbação**: introdução de uma pequena perturbação pontual no centro do domínio, modelada como:

```
int cx = N / 2, cy = N / 2, cz = N / 2;
u[cx][cy][cz] = 0.0f; // inicialmente mas, para pertubação, foi ajustada para 1.0f
```

Essa perturbação serve como impulso inicial para observar a dissipação causada pela viscosidade.

O código foi paralelizado usando a diretiva *#pragma omp parallel for*, com testes das três principais estratégias de agendamento (*static*, *dynamic*, *guided*) e diferentes tamanhos de *chunk* (1, 2, 4 e 8). Também foi testada a aplicação da cláusula *collapse*(3), permitindo a fusão dos três laços aninhados da iteração espacial:

```
#pragma omp parallel for collapse(3) schedule(dynamic, chunk_size)
for (int i = 1; i < N-1; i++)
    for (int j = 1; j < N-1; j++)
        for (int k = 1; k < N-1; k++)
          // Cálculo do laplaciano e atualização
```

Cada combinação de configuração foi executada e os tempos de execução registrados em um arquivo *CSV* para análise posterior. A cláusula *collapse* foi usada para avaliar sua influência na granularidade e balanceamento de carga.

A simulação com perturbação gerou arquivos binários contendo *snapshots* periódicos do campo de velocidade. Esses dados foram lidos em *Python* com *numpy* e visualizados com *plotly*, usando uma

fatia 2D do plano central do volume para facilitar a análise. A visualização foi configurada com um controle deslizante (*slider*) que permite navegar no tempo, fornecendo uma animação interativa da dissipação da perturbação.

3. RESULTADOS

Os experimentos realizados permitiram avaliar dois aspectos principais da simulação: a estabilidade e difusão do campo de velocidade do fluido e o impacto das diretivas de paralelização *OpenMP* na performance computacional.

Nos testes sem perturbação inicial, o campo de velocidade permaneceu constante ao longo do tempo, indicando que o esquema numérico é estável quando não há gradientes espaciais. Já ao introduzir uma perturbação pontual no centro do domínio, foi observada uma difusão suave e isotrópica da mesma, como esperado de um processo puramente viscoso (sem pressão nem forças externas).

A animação das fatias centrais *2D* mostrou que a perturbação se espalha de maneira simétrica em todas as direções, com intensidade decrescendo gradualmente, característica típica de um decaimento difusivo exponencial.

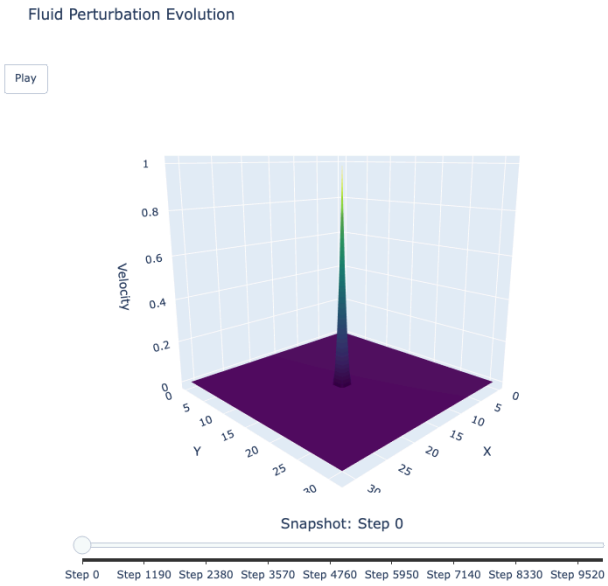

Figure 1: Fatia central do campo de velocidade no início da simulação, destacando a perturbação localizada.

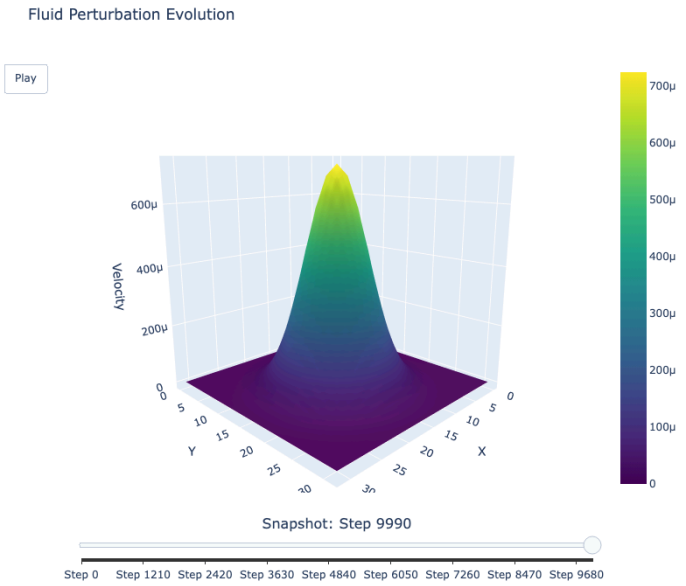

Figure 2: Mesmo corte após *NSTEPS*, evidenciando o espalhamento suave da perturbação (vide escala).

Sem pertubação podemos esperar simplesmente algo como a imagem abaixo.

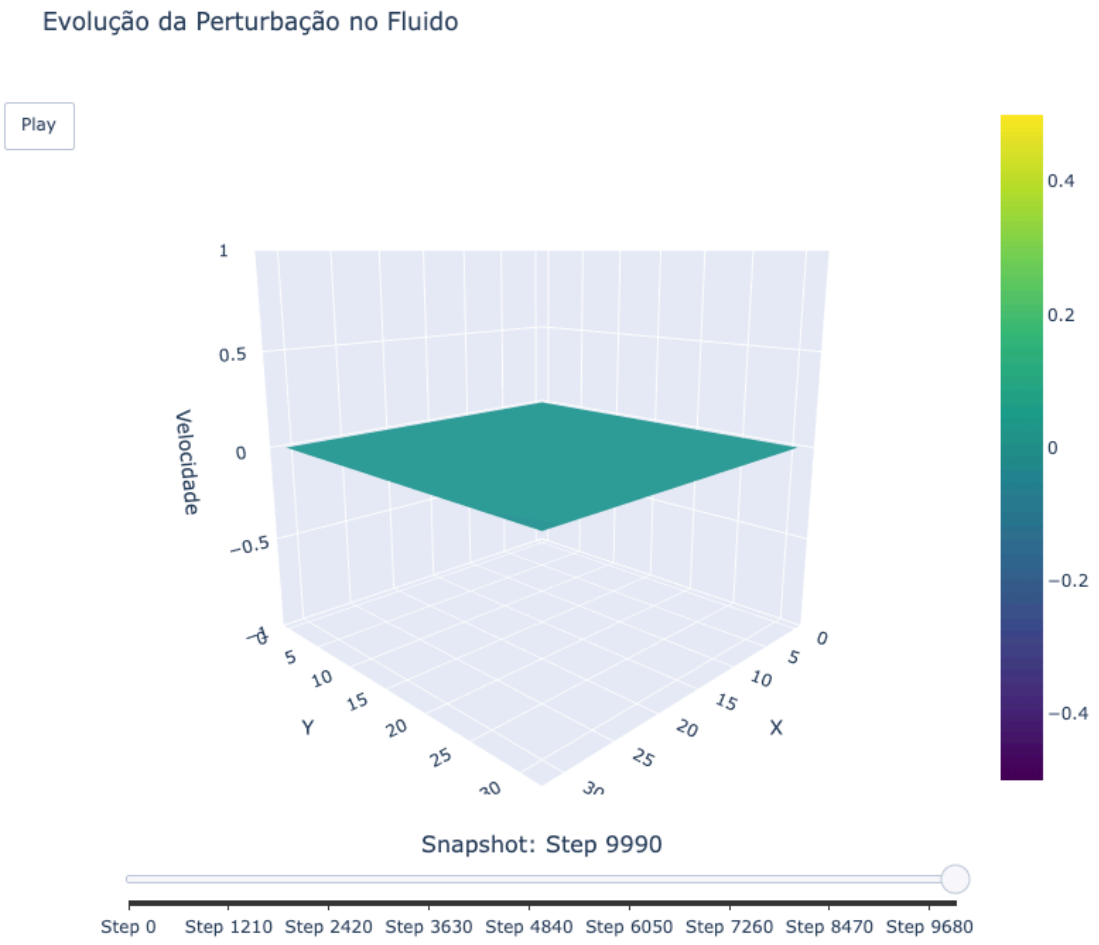

Figure 3: Fatia central do campo de velocidade no final da simulação, sem perturbação.

Os tempos de execução foram extraídos automaticamente e registrados em arquivos *CSV*, tanto para o caso sem perturbação quanto para o com perturbação. A tabela abaixo resume os principais resultados de desempenho, agrupando por estratégia de agendamento (*schedule*), tamanho do *chunk* e uso ou não da cláusula *collapse*.

Schedule Type	Chunk Size	Collapse	Time (s)
static	1	no	0.777193
static	1	yes	0.965302
static	2	no	0.794112
static	2	yes	0.881914
static	4	no	0.778012
static	4	yes	0.855424
static	8	no	0.797010
static	8	yes	0.808329
dynamic	1	no	0.769642
dynamic	1	yes	12.356255
dynamic	2	no	0.749796
dynamic	2	yes	6.748700
dynamic	4	no	0.753910
dynamic	4	yes	3.812959
dynamic	8	no	0.710674
dynamic	8	yes	2.206722
guided	1	no	0.766019
guided	1	yes	0.853232
guided	2	no	0.764410
guided	2	yes	0.844573
guided	4	no	0.761280
guided	4	yes	0.841754
guided	8	no	0.711521
guided	8	yes	0.886514

Tabela 1: Tempo de execução sem perturbação

Schedule Type	Chunk Size	Collapse	Time (s)
static	1	no	1.085867
static	1	yes	1.543973
static	2	no	1.132759
static	2	yes	1.342473
static	4	no	1.075633
static	4	yes	1.207385
static	8	no	1.342248
static	8	yes	1.500057
dynamic	1	no	1.369867
dynamic	1	yes	16.967685
dynamic	2	no	1.079003
dynamic	2	yes	9.025466
dynamic	4	no	1.068436
dynamic	4	yes	4.944012
dynamic	8	no	1.070937
dynamic	8	yes	2.838898
guided	1	no	1.056918
guided	1	yes	1.234031
guided	2	no	1.063896
guided	2	yes	1.172774
guided	4	no	1.099474
guided	4	yes	1.169514
guided	8	no	1.173686
guided	8	yes	1.557259

Tabela 2: Tempo de execução com perturbação

Os dados revelam comportamentos distintos, como esperado, entre as estratégias:

- O agendamento estático foi o mais rápido de forma consistente quando *collapse* não foi usado.
- A cláusula *collapse*(3), embora aumente a granularidade, piora o desempenho nas configurações com agendamento dinâmico, chegando a multiplicar o tempo de execução em até 16x para *chunk_size* = 1.
- Com agendamento guiado, o tempo de execução foi competitivo e estável, mesmo com *collapse*, indicando bom balanceamento de carga e baixa sobrecarga de gerenciamento.

As figuras a seguir ilustram a variação do tempo em função do tipo de agendamento e tamanho de *chunk*, com e sem *collapse*.

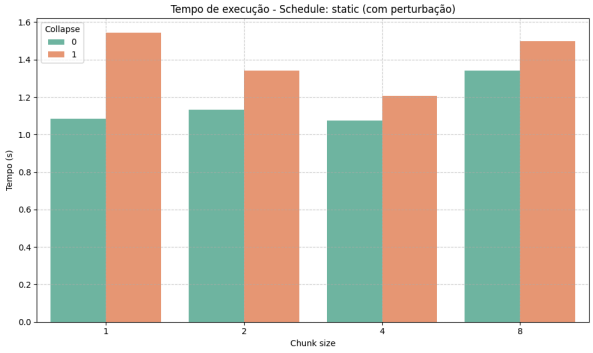

Figure 4: Tempo de execução - *Schedule: static* (com perturbação) Tempo (s) vs. *Chunk Size*

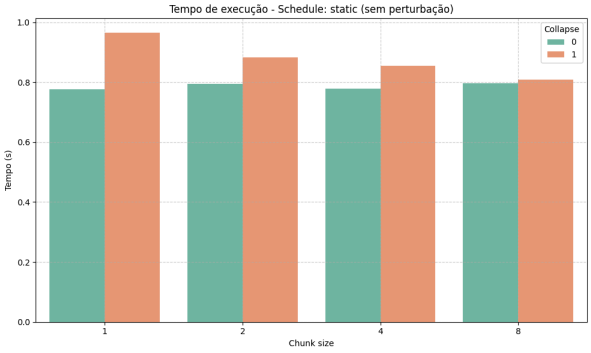

Figure 5: Tempo de execução - *Schedule: static* (sem perturbação) Tempo (s) vs. *Chunk Size*

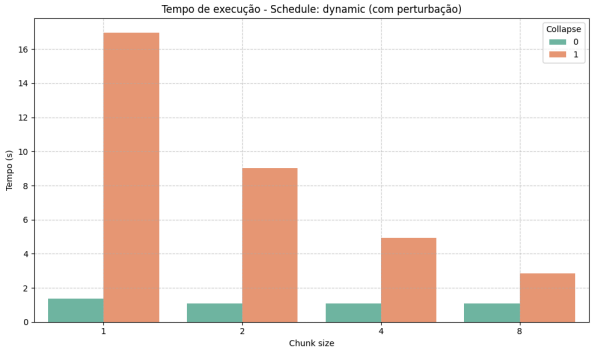

Figure 6: Tempo de execução - *Schedule: dynamic* (com perturbação) Tempo (s) vs. *Chunk Size*

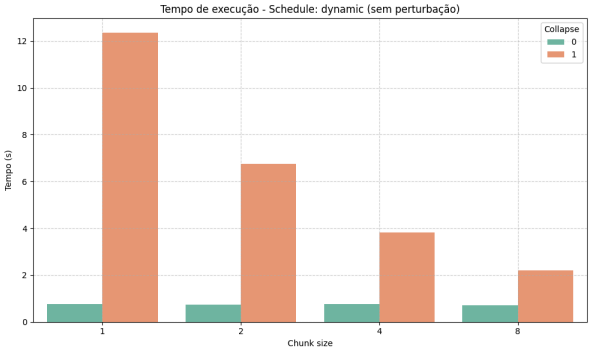

Figure 7: Tempo de execução - *Schedule: dynamic* (sem perturbação) Tempo (s) vs. *Chunk Size*

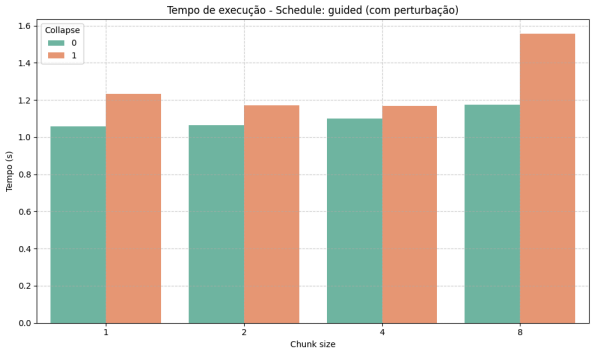

Figure 8: Tempo de execução - *Schedule: guided* (com perturbação) Tempo (s) vs. *Chunk Size*

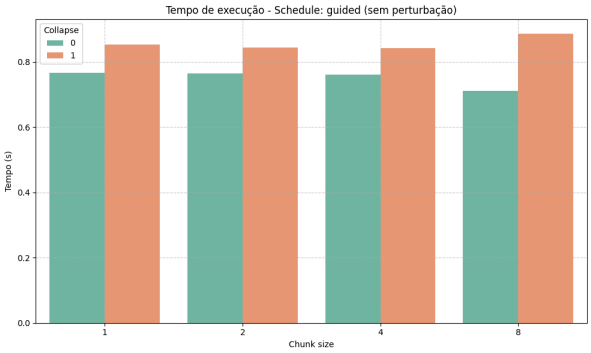

Figure 9: Tempo de execução - *Schedule: guided* (sem perturbação) Tempo (s) vs. *Chunk Size*

O agendamento *static* distribui de maneira fixa e antecipada os *chunks* entre as *threads*, sem considerar o tempo real de execução de cada bloco. Com isso, ele é eficiente em ambientes homogêneos e previsíveis. Sem perturbação, os tempos de execução foram estáveis e próximos entre si, tanto com quanto sem a cláusula *collapse*. O aumento de tempo com *collapse* foi pequeno, indicando um *overhead* moderado devido ao maior número de iterações geradas pelo *loop colapsado*. Com perturbação, houve um aumento geral nos tempos, especialmente com *collapse*. Isso se deve ao fato de que, com uma distribuição fixa de trabalho, qualquer desbalanceamento causado por *threads* interrompidas (ou mais lentas) afeta diretamente o tempo total de execução, uma vez que as demais não podem compensar esse atraso. Assim *static* oferece bom desempenho em condições ideais, mas apresenta limitações sob desbalanceamentos de carga, sendo mais sensível à presença de perturbações.

O agendamento *dynamic* aloca os *chunks* de maneira dinâmica: à medida que uma *thread* termina seu trabalho, ela solicita o próximo *chunk* disponível. Isso permite melhor balanceamento de carga, mas com maior *overhead* de controle e sincronização. Sem perturbação, o desempenho sem *collapse* foi excelente, sendo o mais rápido entre todas as configurações testadas. No entanto, ao usar *collapse*, o desempenho degradou de forma severa, com tempos até *16x* maiores. Isso ocorre porque, ao colapsar dois *loops*, o número de iterações totais aumenta significativamente e o *dynamic* tenta redistribuí-las com granularidade mais fina. O *overhead* de controle torna-se expressivo, especialmente com *chunk_size* pequeno. Com perturbação, o mesmo padrão foi observado, com desempenho aceitável sem *collapse*, mas muito ruim com *collapse*, reforçando a sensibilidade dessa estratégia a *overheads* adicionais e à fragmentação de tarefas. Então o *dynamic* sem collapse pode ser eficiente, mas sua combinação com collapse é fortemente desaconselhada, especialmente em aplicações com alto número de iterações e possíveis desbalanceamentos.

O *guided* funciona de forma híbrida: as primeiras iterações são atribuídas em *chunks* grandes, que vão diminuindo gradualmente de tamanho à medida que mais *threads* se tornam disponíveis. Isso reduz o *overhead* de distribuição (como no *static*) e melhora o balanceamento (como no *dynamic*). Sem perturbação, o desempenho foi estável e consistente em todas as configurações, com tempos semelhantes com e sem *collapse*. Com perturbação, os tempos aumentaram de forma previsível, mas

mantiveram boa estabilidade. O uso de *collapse* não provocou degradação significativa. Isso evidencia que o *guided* foi capaz de lidar eficientemente tanto com a sobrecarga estrutural de *collapse* quanto com variações externas de desempenho. E com isso o *guided* demonstrou ser a melhor estratégia no contexto avaliado, combinando boa distribuição de carga, baixa sensibilidade a perturbações e tolerância à cláusula collapse.

O motivo pelo qual o agendamento *guided* se destacou está diretamente relacionado à sua natureza adaptativa. Ao iniciar com grandes *chunks*, ele reduz o *overhead* de agendamento presente no início da execução. Conforme o número de iterações restantes diminui, ele passa a dividir o trabalho em unidades menores, permitindo melhor adaptação às capacidades reais de cada *thread*, mesmo em ambientes com perturbações.

Essa abordagem é particularmente vantajosa no uso com a cláusula *collapse*, pois a distribuição variável das iterações totais (resultado do *loop* colapsado) é naturalmente ajustada pela estratégia *guided*. Por outro lado, *static* sofre com alocação rígida, e *dynamic* tem *overhead* excessivo, especialmente quando o espaço de iterações é ampliado artificialmente com *collapse*.

4. CONCLUSÃO

Neste trabalho, investigamos o impacto das estratégias de agendamento (*static*, *dynamic*, *guided*) e da cláusula *collapse* no desempenho de uma aplicação paralela implementada com *OpenMP*, considerando cenários com e sem perturbações externas.

Os experimentos demonstraram que a escolha da estratégia de agendamento influencia significativamente a eficiência da paralelização, especialmente quando há desbalanceamentos de carga ou quando se utiliza a cláusula *collapse*, que aumenta o número total de iterações.

O agendamento *static* mostrou-se eficiente em ambientes estáveis e previsíveis, mas sensível a variações externas. Já o agendamento *dynamic*, apesar de oferecer bom balanceamento de carga em situações simples, apresentou um custo elevado de *overhead* quando utilizado com *collapse*, tornando-o ineficiente nesse contexto. Por fim, o agendamento *guided* combinou o melhor dos dois mundos: apresentou robustez frente à perturbação, bom balanceamento de carga e desempenho consistente, mesmo com o uso da cláusula *collapse*.

Concluímos, portanto, que para aplicações com loops aninhados e suscetíveis a interferências externas, o agendamento *guided* com *collapse* representa a escolha mais equilibrada entre desempenho e robustez, oferecendo uma solução adaptável e eficiente para cenários reais de execução paralela.

5. ANEXOS

- Repositório no Github com o programa desenvolvido: https://github.com/ErnaneJ/parallel-programming-dca3703