



## Tarefa 11: Impacto das cláusulas *Schedule* e *Collapse*

Discente: Quélita Míriam  
Docente: Samuel Xavier de Souza

### I. INTRODUÇÃO

Neste relatório, avalia-se o impacto das cláusulas *schedule* e *collapse* sobre o desempenho de uma simulação numérica complexa, como as baseadas nas equações de Navier-Stokes. A análise visa compreender como essas diretivas influenciam o tempo de execução da aplicação.

Neste trabalho, foi desenvolvida uma simulação numérica do movimento de um fluido tridimensional ao longo do tempo utilizando a equação de Navier-Stokes simplificada, considerando apenas os efeitos da viscosidade. Para facilitar a implementação, foram desconsideradas as componentes de pressão e quaisquer forças externas. A discretização do domínio foi feita usando o método de diferenças finitas, com espaçamento uniforme no espaço e passos de tempo pequenos para garantir a estabilidade da simulação. A simulação foi feita em um cubo tridimensional de tamanho fixo, com o fluido inicialmente em repouso, exceto por uma pequena perturbação localizada no centro do domínio. Essa perturbação serviu para testar se o campo de velocidades se comportaria de forma esperada, ou seja, se a energia inicial se dissiparia suavemente com o tempo devido à viscosidade.

### II. METODOLOGIA

Foi utilizado um código em C com OpenMP que simula a propagação de uma perturbação em um fluido tridimensional, aplicando a equação de difusão por diferenças finitas. A malha utilizada é de dimensões  $50 \times 50 \times 50$  e a simulação ocorre por 100 passos de tempo.

A metodologia envolveu primeiramente a implementação da simulação de forma sequencial, com a atualização iterativa do campo de velocidades utilizando a equação de difusão. Em seguida, o código foi paralelizado com OpenMP, permitindo testar diferentes estratégias de distribuição de trabalho entre os núcleos do processador. Foram utilizadas três formas de escalonamento (*schedule*) para dividir os laços de iteração entre as threads: *static*, *dynamic* e *guided*. Além disso, foi explorada a diretiva *collapse*, que paraleliza mais de um laço aninhado de forma combinada, aumentando o grau de paralelismo. Diversos tamanhos de blocos de iteração (*chunks*) também foram testados.

Ao usar o OpenMP para paralelizar loops, a cláusula *schedule* define como as iterações do laço serão divididas entre as threads. Cada uma dessas estratégias tem um comportamento diferente:

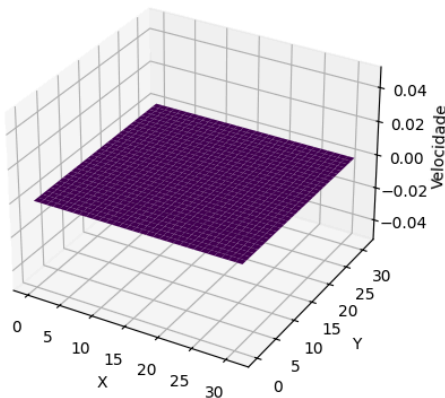
- **Static:** divide as iterações do loop em partes iguais entre as threads, de forma fixa e antecipada, antes do loop começar.
- **Dinamic:** as iterações são divididas em blocos menores (*chunks*), e as threads pegam um novo bloco assim que terminam o anterior.
- **Guided:** os primeiros blocos são maiores e vão diminuindo de tamanho conforme a execução avança.



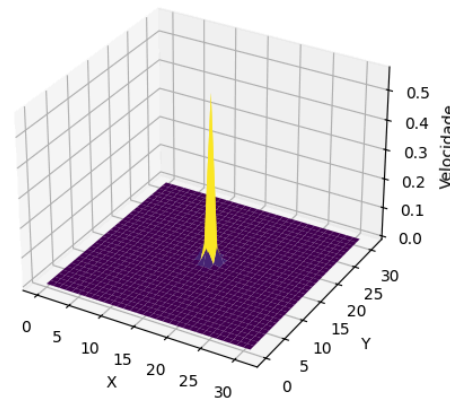
### III. RESULTADOS E DISCUSSÕES

Abaixo, os gráficos representam os campos de velocidades iniciais do fluido antes e após qualquer perturbação. Cada ponto da superfície mostra a velocidade do fluido em uma posição (x, y) da malha, sendo o eixo Z a indicação da intensidade da velocidade em cada ponto da grade. Apresentando desempenho esperado para a simulação de movimento de um fluido.

Campo de Velocidade - Fluido Parado



Campo de Velocidade - Após Perturbação



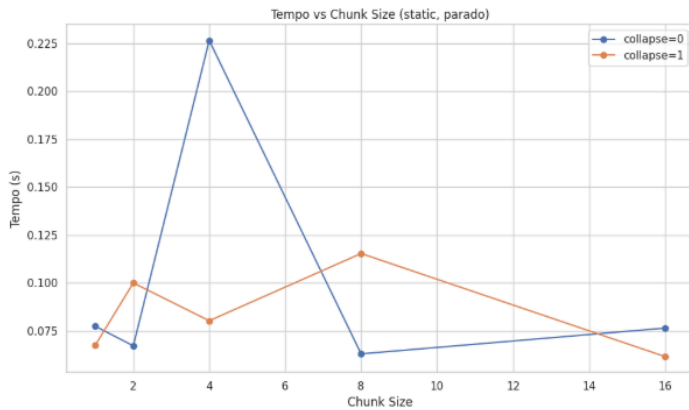
Os resultados obtidos demonstraram comportamentos distintos para cada tipo de escalonamento. O escalonamento *static* divide as iterações igualmente entre as threads no início da execução, sendo mais eficiente quando a carga de trabalho é equilibrada e previsível. No entanto, isso pode não ser ideal quando há variações no custo das iterações.

Seu desempenho foi estável e bom com *collapse=0*, enquanto que com *collapse=1* o tempo oscilou bastante, parecendo ineficiente, provavelmente porque a sobrecarga de colapsar dois loops paralelos não compensa com essa configuração.

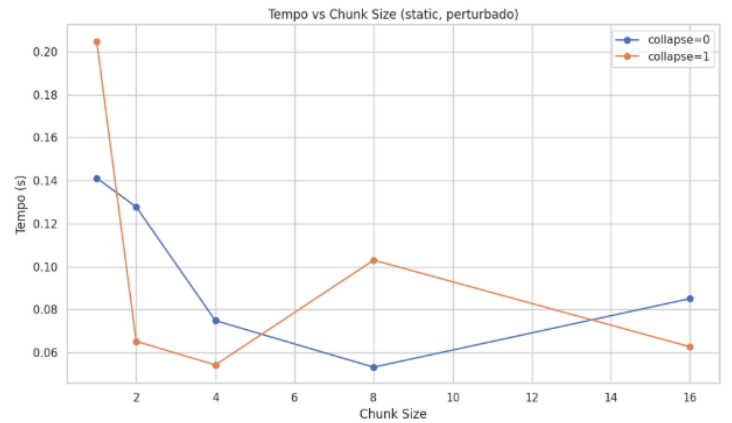


Universidade Federal do Rio Grande do Norte - UFRN  
Departamento de Engenharia da Computação e Automação - DCA  
DCA3703 - Programação Paralela

Static - Fluido parado



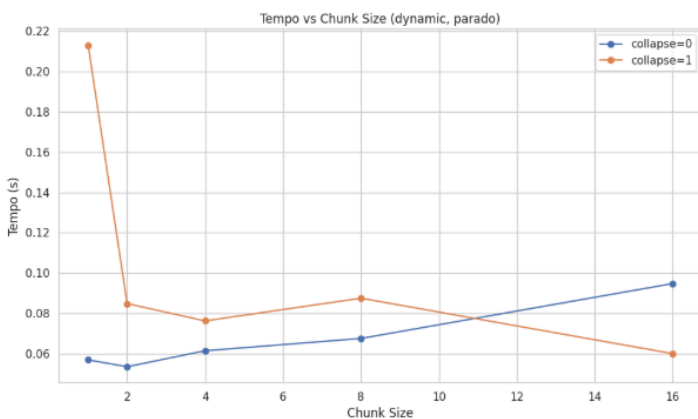
Static - Fluido perturbado



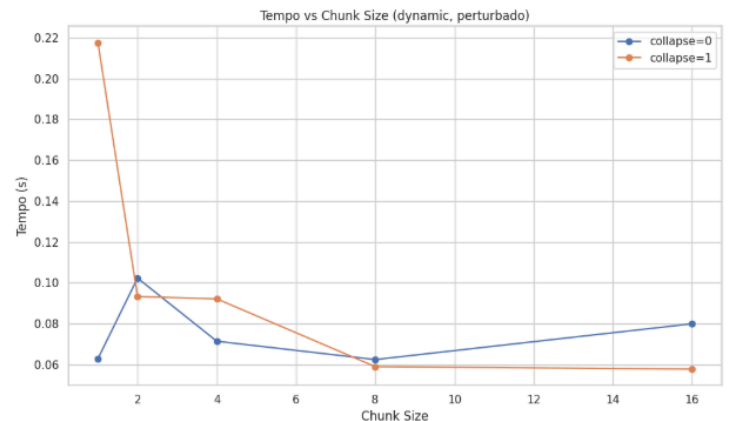
Já o **dynamic** atribui os blocos de iteração às *threads* conforme elas ficam disponíveis, o que é vantajoso quando a carga de trabalho é irregular, pois ajuda a manter todas as *threads* ocupadas. Os tempos com dynamic foram consistentemente baixos com ou sem collapse, mostrando boa eficiência.

O *dynamic* teve bom desempenho para *chunks* pequenos e sem *collapse*. O *collapse=1* parece piorar o desempenho, possivelmente por conflito com a alocação dinâmica e maior sobrecarga de balanceamento.

Dynamic - Fluido parado



Dynamic - Fluido perturbado



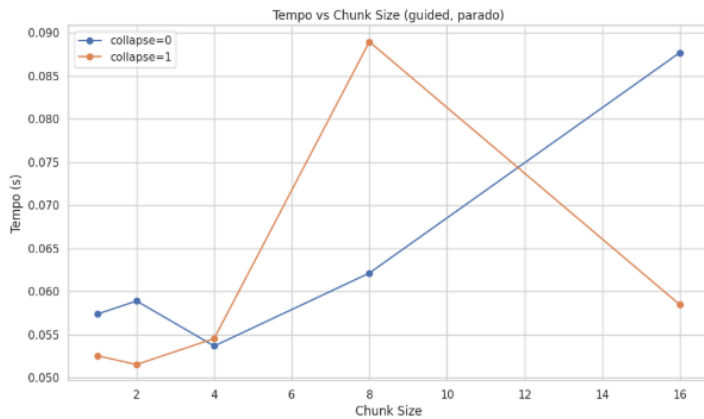
Por fim, o **guided** é uma abordagem intermediária, onde blocos maiores são atribuídos inicialmente e vão diminuindo com o tempo. Isso combina o balanceamento de carga com a redução de *overhead*, e apresentou os melhores tempos de execução entre todas as estratégias.

O *guided* é eficiente quando bem configurado, mas muito sensível a *chunk size*, especialmente com *collapse=1*. Foi ideal para a simulação de perturbação.

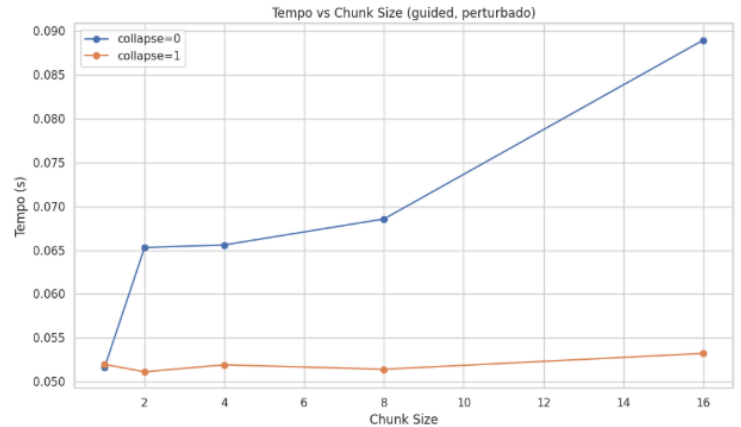


Universidade Federal do Rio Grande do Norte - UFRN  
Departamento de Engenharia da Computação e Automação - DCA  
DCA3703 - Programação Paralela

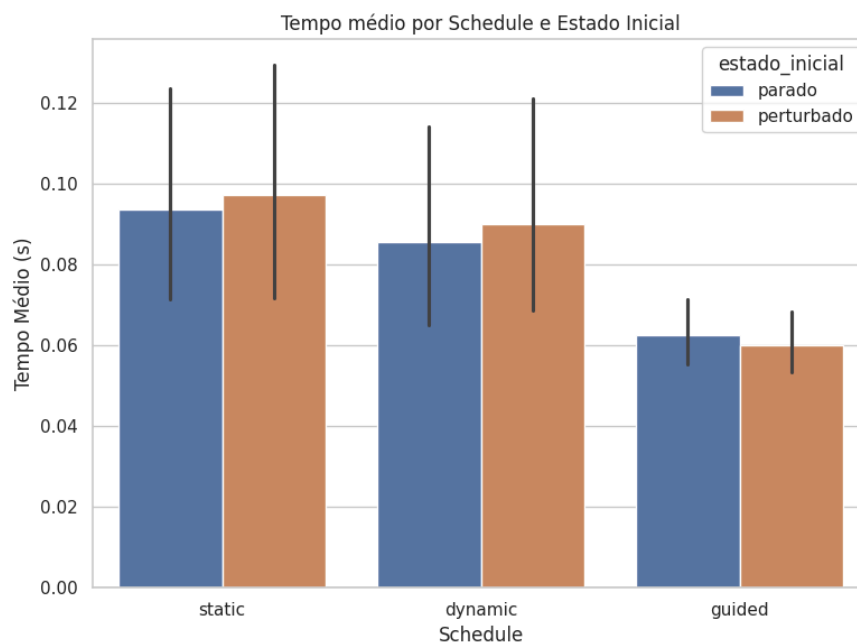
Guided - Fluido parado



Guided - Fluido perturbado



O gráfico abaixo representa os *schedules* usados e seu tempo médio de execução. As linhas pretas são linhas de erro (*error bars*), indicando a incerteza ou variação dos dados. As linhas pequenas representam medições consistentes, enquanto linhas grandes medições instáveis ou com muita flutuação.



Quando o fluido está parado, geralmente há menos trabalho computacional a ser feito, e isso pode ser percebido nos tempos mais baixos na maioria das execuções com esse estado. Já no estado perturbado, é comum observar tempos um pouco maiores, pois há mais variações e cálculos associados à propagação do movimento no fluido.

A variação dos tempos com diferentes tamanhos de *chunk* mostra que nem sempre um *chunk* maior resulta em melhor desempenho. Isso é esperado, pois *chunks* muito grandes podem sobrecarregar



Universidade Federal do Rio Grande do Norte - UFRN  
Departamento de Engenharia da Computação e Automação - DCA  
DCA3703 - Programação Paralela

algumas *threads* e causar desequilíbrio na carga, enquanto *chunks* muito pequenos geram sobrecarga na criação e gerenciamento de tarefas. O comportamento ideal depende do tipo de schedule usado.

#### IV. CONCLUSÃO

Conclui-se que a paralelização do código trouxe benefícios significativos de desempenho, especialmente com o uso do escalonamento guided e da cláusula collapse. O trabalho demonstrou, de forma prática, como diferentes estratégias de distribuição de tarefas impactam no desempenho de simulações científicas, e reforça a importância de uma escolha adequada de parâmetros de paralelismo para explorar ao máximo os recursos computacionais disponíveis.