Universidade Federal do Rio Grande do Norte
Departamento de Engenharia da Computação e Automação
DCA3703 - Programação Paralela
Tarefa 13: Afinidade de threads
Aluno: Daniel Bruno Trindade da Silva

1 Introdução

Este relatório tem como objetivo apresentar os conhecimentos adquiridos durante a realização da Tarefa 13 da disciplina de **Computação Paralela**. A atividade consistiu em avaliar a escalabilidade do programa desenvolvido na Tarefa 11 — um simulador da velocidade de um fluido utilizando a equação de Navier-Stokes — aplicando diferentes políticas de afinidade de *threads*.

2 Enunciado

Avalie como a escalabilidade do seu código de Navier-Stokes muda ao utilizar os diversos tipos de afinidades de *threads* suportados pelo sistema operacional e pelo OpenMP, no mesmo nó de computação do NPAD utilizado para a Tarefa 12.

3 Desenvolvimento

Na Tarefa 11, desenvolvemos duas versões de um programa para simular a velocidade de um fluido: uma versão sequencial (serial) e outra paralelizada com OpenMP. Para a análise requerida nesta tarefa, utilizamos a versão paralelizada do código.

Nesta tarefa, analisamos os impactos da cláusula `proc_bind()` com as seguintes políticas de afinidade:

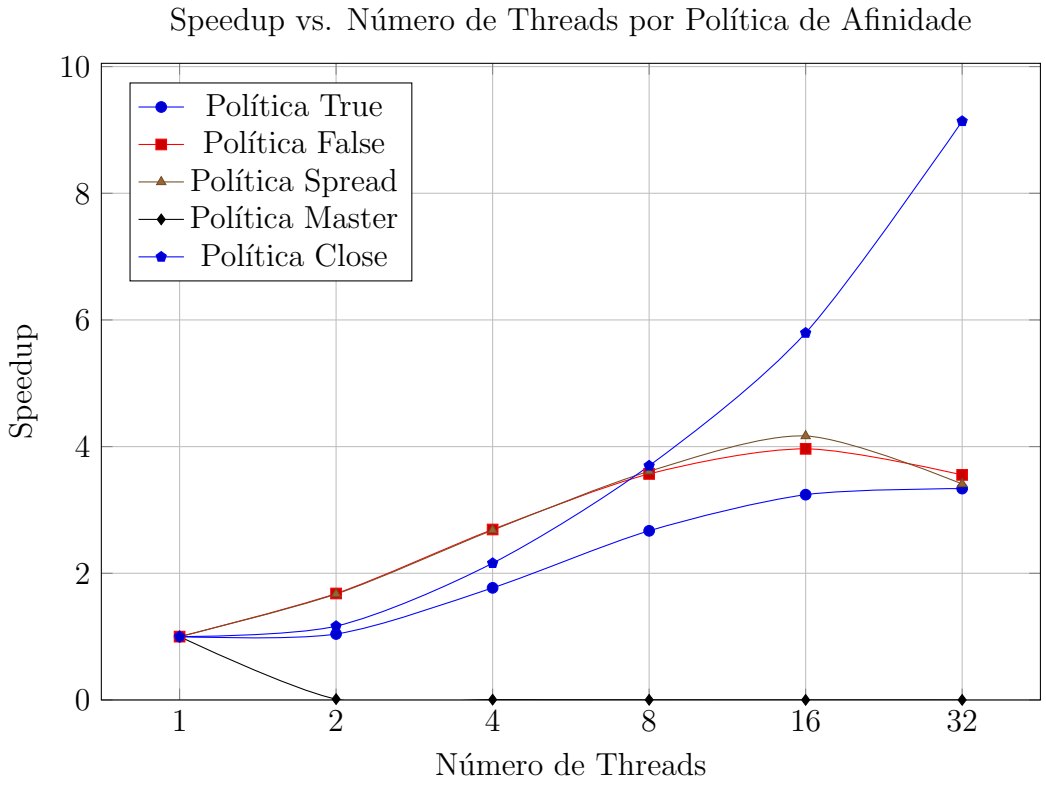
- `spread` — O openMP distribui as *threads* de forma espalhada pelos processadores, maximizando a distância entre elas. O objetivo é utilizar o máximo de recursos do hardware possível, como diferentes *sockets* ou núcleos físicos. Aumenta a latência de comunicação entre as *threads* em troca de menos disputa por recurso. Ideal quando temos programas com *threads* independentes;
- `close` — O openMP agrupa as *threads* próximas umas das outras, preferencialmente no mesmo *socket* ou núcleos adjacentes ao da *thread* master. Utilizada quando queremos manter as *threads* próximas pra diminuir a latência, mas não queremos sobrecarregar um mesmo núcleo;
- `master` — Nessa política todas as *threads* são alocadas no mesmo local onde a *thread* principal (*master/primary*) está executando. Utilizado quando a comunicação entre *threads* é muito intensa e isso compença a perda de paralelismo. Pois ela gera disputa por recursos, mas reduz a latência de comunicação entre as *threads*;
- `true` — herda a política de afinidade da região paralela pai. Se não houver região pai, comporta-se de acordo com a política padrão definida pela implementação.
- `false` — Desativa a afinidade de *threads* o escalonador do sistema operacional é quem vai decidir onde cada *thread* ficará. Com isso pode haver migração de *threads* entre núcleos causando mais overhead e menos previsibilidade;

Reorganizamos o código de forma a possibilitar o teste de todas as políticas em uma única execução. Assim como realizado na Tarefa 12, o código foi executado com 1, 2, 4, 8, 16 e 32 *threads*, para que ao final pudéssemos analisar se houve influência dessas políticas de afinidade na eficiência do código.

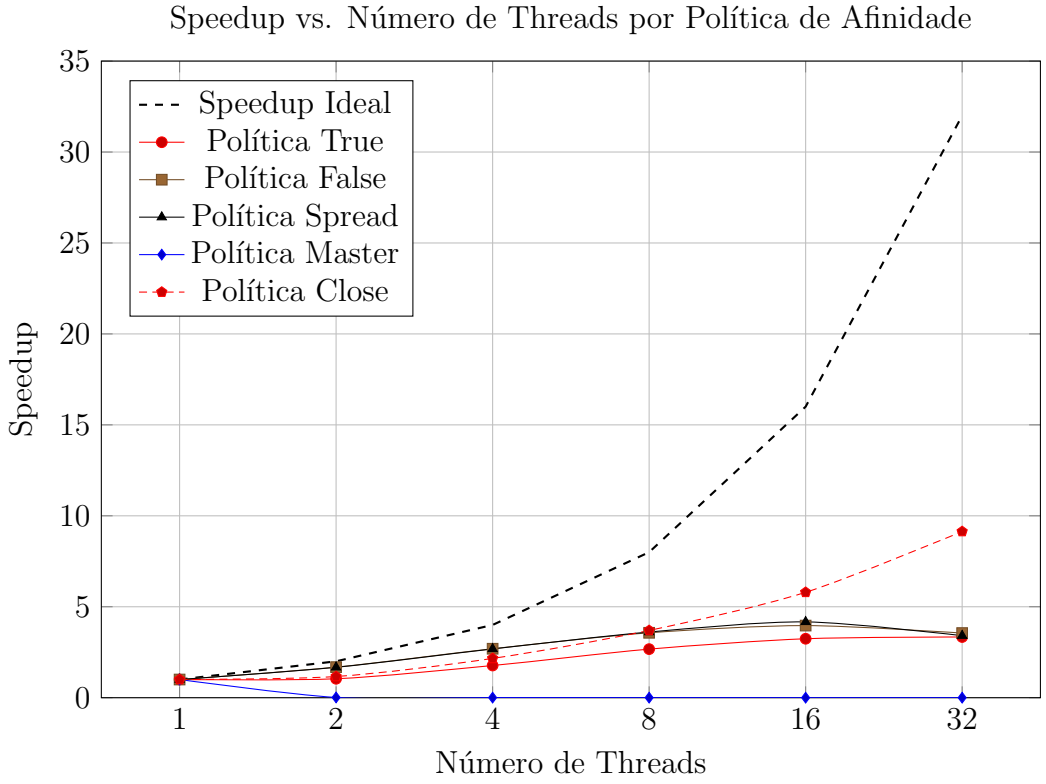
O código foi executado no super computador da universidade utilizando o nó com o processador intel-128, cada teste foi executado 6 vezes para termos a certeza da constância dos resultados.

4 Resultados

Iniciaremos avaliando a escalabilidade forte fixando o tamanho de nosso problema e aumentando o número de threads utilizados para o processamento. A figura abaixo mostra nossos resultados:



Na figura abaixo temos o mesmo gráfico, mas com a curva do *speedup* ideal para fins de comparação



Para a analise da escalabilidade fraca fixamos o poder computacional no uso de 8 threads e aumentamos o tamanho do problema gradualmente em potencias de 2. Assim podemos verificar qual das políticas melhor se aproximou da eficiência ideal que tem valor igual a 1.

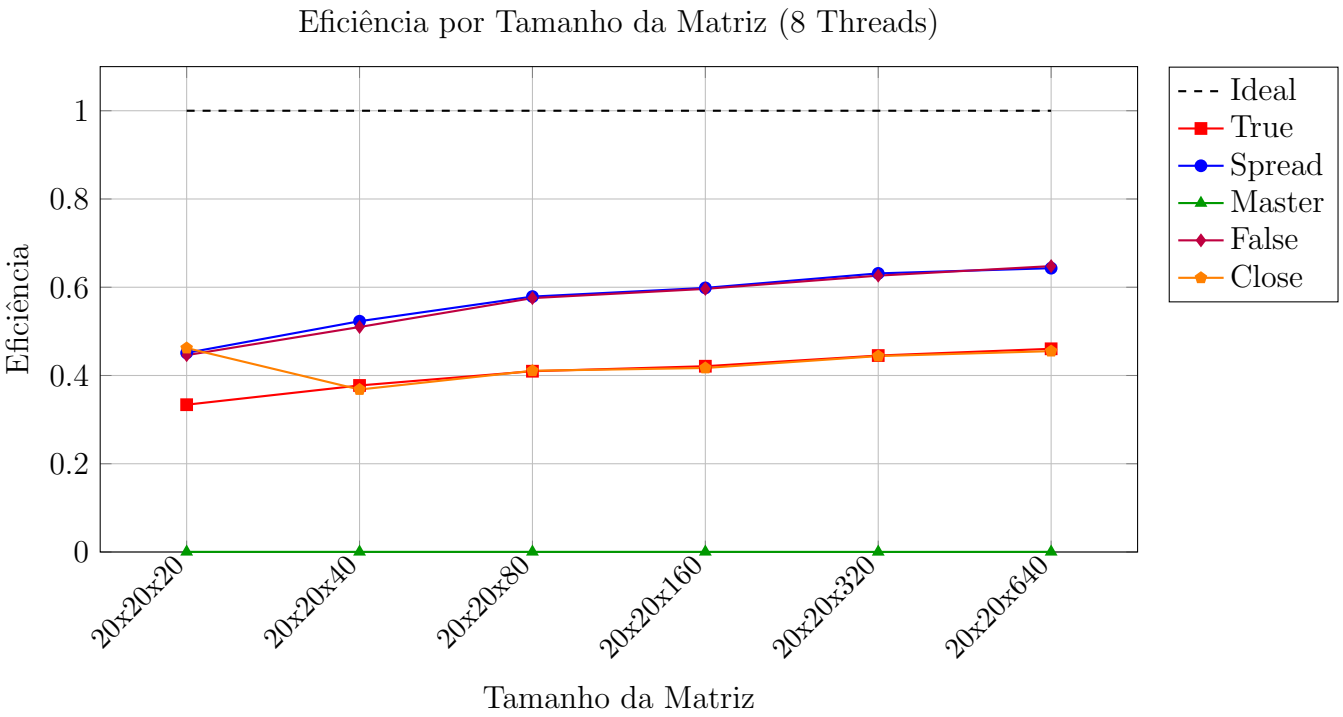


Figure 1: Comparação da Eficiência de Políticas de Afinidade com 8 Threads

5 Analise dos Resultados

A análise dos resultados obtidos nos permite inferir sobre o comportamento das diferentes políticas de afinidade de threads em dois cenários distintos: escalabilidade forte e escalabilidade fraca.

5.1 Escala Forte: Speedup vs. Número de Threads

Observando os gráficos de Speedup em função do número de threads, podemos notar comportamentos bastante distintos entre as políticas:

- **Política Master:** Esta política apresentou o pior desempenho, com um speedup que decai drasticamente com o aumento do número de threads, chegando a valores próximos de zero. Isso indica que forçar todas as threads a executarem no mesmo local da thread principal gerou uma contenção de recursos tão severa que o overhead da paralelização superou qualquer ganho, tornando o programa mais lento do que a versão sequencial em muitos casos.

- **Política True:** A política true demonstrou um speedup modesto, com um crescimento inicial que rapidamente atinge um platô. O speedup máximo parece ser limitado a aproximadamente 3.3x, mesmo com 32 threads. Isso sugere que a política de afinidade herdada ou padrão não está otimizando eficientemente a distribuição das threads para este código, levando a um baixo aproveitamento dos recursos computacionais adicionais.

- **Políticas False e Spread:** Ambas as políticas apresentaram um comportamento semelhante, tem o pico de speedup entre 8 e 16 threads e depois observa-se uma queda ao utilizar 32 threads. Isso pode indicar que, com um número muito alto de threads, o custo de comunicação ou a disputa por recursos distribuídos (no caso da spread) ou a migração de threads (no caso da false) começa a superar os benefícios do paralelismo adicional para o tamanho do problema fixado.

- **Política Close:** Esta política se destacou por apresentar o melhor speedup com um número maior de threads, chegando a mais de 9x com 32 threads. Isso sugere que agrupar as threads

próximas, minimizando a latência de comunicação, foi benéfico para este problema, especialmente quando mais threads são empregadas. O crescimento do speedup parece mais sustentado com o aumento das threads em comparação com false e spread, embora não se aproxime do ideal.

5.2 Escala Fraca: Eficiência vs. Tamanho da Matriz (com 8 Threads)

Analisando o gráfico de eficiência para a escalabilidade fraca, onde o tamanho do problema por thread é mantido constante (fixando 8 threads e aumentando o tamanho total da matriz):

- **Política Master:** Consistentemente, a política master apresentou uma eficiência próxima de zero para todos os tamanhos de matriz. Isso reforça a conclusão da análise de escala forte: esta política é inadequada para este problema, causando severa degradação de desempenho.
- **Políticas Spread e False:** Estas duas políticas foram as que apresentaram a maior eficiência entre as testadas, com valores variando aproximadamente entre 0.45 e 0.65. Ambas demonstram uma tendência de leve aumento da eficiência com o aumento do tamanho da matriz. Isso sugere que, para 8 threads, espalhar as threads (spread) ou deixar o sistema operacional gerenciá-las (false) permite uma utilização razoável dos recursos à medida que o problema cresce. A política false teve uma ligeira vantagem na maioria dos tamanhos de matriz, atingindo a maior eficiência (aproximadamente 0.648).
- **Política True:** A política true mostrou uma eficiência moderada, começando em torno de 0.33 e aumentando gradualmente até aproximadamente 0.46. Embora haja uma melhoria com o aumento do tamanho do problema, ela permanece significativamente abaixo das políticas spread e false.
- **Política Close:** A política close apresentou um comportamento interessante. Começou com uma eficiência relativamente alta (aproximadamente 0.46 para a menor matriz), similar à true para a maior matriz. No entanto, para o segundo tamanho de matriz (20x20x40), sua eficiência caiu para cerca de 0.37, antes de retomar uma tendência de crescimento gradual com o aumento do problema, alcançando aproximadamente 0.45 para a maior matriz. Essa queda inicial pode indicar que, para problemas menores com 8 threads, agrupar as threads muito próximas pode não ser tão vantajoso, possivelmente devido à sobreposição de caches ou outros recursos locais, antes que o aumento do trabalho por thread compense essa proximidade.

6 Conclusão

A realização desta tarefa permitiu uma análise aprofundada do impacto das políticas de afinidade de *threads* na escalabilidade de um simulador da equação de Navier-Stokes, evidenciando que a escolha da política exerce influência considerável no desempenho e não é trivial. Os experimentos demonstraram que, enquanto a política `master` consistentemente degradou o desempenho, as políticas `false` e `spread` mostraram-se eficazes para contagens intermediárias de *threads* na escalabilidade forte e foram as mais eficientes na escalabilidade fraca com 8 *threads*. Notavelmente, a política `close` apresentou o melhor *speedup* com um número elevado de *threads* (32) na escalabilidade forte, sublinhando que a configuração ótima depende das características da aplicação, do volume de dados e do número de *threads*. Nenhuma política atingiu o desempenho ideal, indicando limitações inerentes à paralelização do código ou à arquitetura. Conclui-se que a atividade foi fundamental para consolidar conhecimentos práticos sobre a otimização e avaliação de desempenho em computação paralela, reforçando a necessidade de experimentação para a escolha da política de afinidade mais adequada.