

Sincronização Orientada por Dependências para Stencils: Modelagem Causal e Avaliação de Barreiras, Semáforos e MPI

Frederico L. Cabral^{1,3}, Carla Osthoff¹, Diego Brandão³, Sanderson L. Gonzaga de Oliveira²

¹Laboratório Nacional de Computação Científica (LNCC)
Petrópolis – RJ – Brasil

²Universidade Federal de São Paulo (UNIFESP)
São José dos Campos – SP – Brasil

³Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (CEFET/RJ)
Rio de Janeiro – RJ – Brasil

frederico.cabral@cefet-rj, osthoff@lncc.br

diego.brandao@cefet-rj.br, sanderson.oliveira@unifesp.br

Resumo. *Barreiras de fase são simples e seguras, mas impõem uma ordem coletiva mais forte que a exigida pelo grafo de dependências de um stencil. Neste artigo, é formalizada a sincronização orientada por dependências para stencils de raio um, tendo como estudo de caso uma decomposição em faixas do Odd-Even Hopscotch para a equação do calor 2D. Um modelo causal distingue ordem de programa, sincronização entre vizinhos e ordenação causada por barreiras, mostrando que um stencil de raio um requer precedência apenas entre interfaces adjacentes. Com base nisso, propõe-se um protocolo OpenMP sem barreiras globais, usando semáforos POSIX direcionais, e compara-se sua localidade ao MPI ponto a ponto com troca de halos. Um modelo de primeira ordem do makespan, isto é, do tempo total necessário para concluir a execução, prevê vantagem dos semáforos quando o custo da espera coletiva imposto pela barreira supera o custo adicional do bloqueio local e do desequilíbrio entre vizinhos. A avaliação, com malha de tamanho equivalente a memória local de um ambiente computacional de última geração contendo 192 núcleos, mantém fixo o arranjo de memória das variantes OpenMP. Entre 32 e 192 threads, os semáforos superam as barreiras em todas as configurações; em 192 threads, reduzem o tempo mediano de forma significativa para `TILE` equivalente ao tamanho da memória cache, onde o MPI puro ainda é o mais rápido nos mesmos testes. Os resultados do estudo de caso sustentam o modelo causal e evidenciam que a sincronização local reduz a espera coletiva, aproximando-se do desempenho do MPI sem reproduzir seu modelo explícito de comunicação.*

1. Introdução

O aumento do número de núcleos por nó tornou a sincronização uma parte cada vez mais visível do custo de aplicações científicas paralelas. Em códigos iterativos baseados em diferenças finitas, uma fase pode executar poucas operações por ponto e, ainda assim, terminar em um ponto de encontro coletivo. Quando isso ocorre repetidamente, diferenças locais de tempo de execução passam a ser propagadas para todo o conjunto de *threads*, e o custo de coordenação deixa de ser secundário [Hager and Wellein 2010, Datta et al. 2008].

Em OpenMP, uma construção de compartilhamento de trabalho normalmente termina com uma barreira implícita, salvo quando a semântica da diretiva permite e o programador solicita sua remoção [OpenMP Architecture Review Board 2021]. A barreira oferece uma garantia forte: nenhuma *thread* inicia a etapa seguinte até que todas tenham concluído a atual. Entretanto, garantia forte não significa dependência necessária. Em uma decomposição espacial de um *stencil* de raio unitário, uma faixa do domínio consome dados apenas de si mesma e das faixas adjacentes. Uma *thread* que opera no topo do domínio não depende diretamente da conclusão de uma faixa distante na parte inferior.

Essa diferença entre *ordem necessária* e *ordem imposta* é o ponto de partida deste trabalho. A hipótese é que, quando a dependência algorítmica é local, uma sincronização vizinho-a-vizinho pode preservar a correção com menor interferência no caminho crítico. Semáforos POSIX são particularmente adequados para testar essa hipótese porque permitem que uma *thread* aguarde explicitamente apenas os produtores dos dados de que necessita, sem um *join* coletivo. MPI fornece um contraponto natural: sob decomposição de domínio, a atualização de halos também coordena apenas processos vizinhos, mas com espaços de endereçamento separados e transferência explícita de dados [MPI Forum 2023, Gropp et al. 2014].

Usamos o método Odd-Even Hopscotch (OEH) para a equação do calor bidimensional como estudo de caso. O OEH é particularmente adequado para esta investigação porque suas quatro fases por passo temporal tornam explícitas as dependências locais associadas ao stencil de cinco pontos [Gourlay 1970, ten Thije Boonkkamp and Verwer 1987]. Essa estrutura permite estabelecer uma correspondência direta entre o grafo de dependências numéricas, as precedências impostas pelo mecanismo de sincronização e o tempo de absoluto observado. Assim, o OEH não é tratado como o objeto exclusivo da contribuição, mas como um caso representativo para investigar sincronização orientada por dependências. As contribuições deste estudo são descritas a seguir.

1. Uma formulação causal da sincronização de stencils, distinguindo as precedências impostas pelo grafo de dependências numéricas da ordenação adicional introduzida por barreiras globais.
2. Um protocolo de sincronização local baseado em créditos direcionais entre faixas vizinhas, implementado com semáforos POSIX, acompanhado de argumentos de correção e de ausência de deadlock.
3. Um modelo analítico de custo que caracteriza o regime em que a redução da espera coletiva compensa o custo adicional da sincronização local e do desequilíbrio entre vizinhos.
4. Uma avaliação experimental controlada comparando OpenMP com barreiras, OpenMP com semáforos e MPI puro, em três granularidades de *tiling* e até 192 *workers*.
5. A caracterização experimental do regime de alta concorrência, no qual a sincronização local supera sistematicamente a barreira global, embora o MPI permaneça como referência de menor tempo absoluto em parte relevante do espaço experimental.

O restante do artigo é organizado da seguinte forma. A Seção 2 posiciona a proposta em relação a trabalhos sobre barreiras, *stencils* e programação MPI/OpenMP. Em seguida, a Seção 3 apresenta o OEH e explicita suas dependências locais; a Seção 4 desenvolve a modelagem causal e de custo; a Seção 5 descreve as implementações avaliadas; a Seção 6 detalha a metodologia experimental; e a Seção 7 analisa os resultados. Por fim, as Seções 8 e 9 apresentam a discussão, as limitações e as conclusões.

2. Trabalhos relacionados

A literatura de sincronização em memória compartilhada inclui algoritmos de barreira voltados à redução de contenção e à escalabilidade do encontro coletivo [Hensgen et al. 1988, Mellor-Crummey and Scott 1991]. Essas técnicas aperfeiçoam a implementação do encontro entre participantes, mas preservam sua semântica global: a fase seguinte só pode começar depois da chegada de todos. Em OpenMP, essa ordenação também aparece implicitamente em várias construções de compartilhamento de trabalho [OpenMP Architecture Review Board 2021]. Assim, reduzir o custo de uma barreira e verificar se ela é necessária são problemas diferentes; este artigo se concentra no segundo.

Em códigos de *stencil*, desempenho e escalabilidade dependem fortemente de localidade, largura de banda, *blocking*, hierarquia de memória e granularidade [Datta et al. 2008, Hager and Wellein 2010]. Ao mesmo tempo, a geometria do stencil determina quais partições realmente trocam dados. Para raio um, apenas subdomínios adjacentes participam de uma dependência entre fronteiras. Os trabalhos clássicos sobre Hopscotch e Odd-Even Hopscotch descrevem a organização numérica em fases [Gourlay 1970, ten Thije Boonkkamp and Verwer 1987]; aqui, essa estrutura é usada para relacionar as dependências que atravessam as faixas ao mecanismo de sincronização que deve preservá-las.

Comparações entre MPI e OpenMP mostram que decomposição, hierarquia de memória e custos de comunicação e sincronização influenciam o desempenho relativo [Rabenseifner et al. 2009, Cabral et al. 2020]. Em decomposições espaciais como a considerada neste trabalho, MPI permite materializar essa localidade por trocas ponto a ponto de halos entre vizinhos [MPI Forum 2023, Gropp et al. 2014, Cabral et al. 2020]. Em memória compartilhada, semáforos POSIX podem representar a mesma precedência sem transferência explícita dos dados [IEEE and The Open Group 2018]. Essa leitura é compatível com a noção de ordem causal de Lamport, na qual relações de precedência definem uma ordem parcial entre eventos [Lamport 1978]. Nosso foco, portanto, não é propor um novo algoritmo de barreira ou um novo mecanismo de comunicação, mas fazer corresponder o grafo de dependências do método numérico ao grafo de sincronização da implementação. A próxima seção torna essa correspondência concreta para o OEH.

3. Odd-Even Hopscotch e dependências locais

Considere a equação do calor bidimensional

$$\frac{\partial u}{\partial t} = \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right), \quad (1)$$

em uma malha uniforme. Para um ponto interno (i, j) , o Laplaciano discreto de cinco pontos depende apenas de $(i \pm 1, j)$ e $(i, j \pm 1)$. Denotando por U a aproximação discreta, uma atualização explícita típica contém o termo

$$\Delta_h U_{i,j} = U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1} - 4U_{i,j}. \quad (2)$$

O OEH alterna duas paridades da malha em dois meios-passos, totalizando quatro fases por passo temporal [Gourlay 1970, ten Thije Boonkkamp and Verwer 1987]. Em

cada fase, parte dos pontos é atualizada e a fase seguinte pode consumir valores recém-produzidos. Para a discussão de sincronização, a propriedade essencial é que as dependências cruzando uma partição espacial permanecem confinadas às fronteiras de subdomínios adjacentes.

Particionamos as $N - 2$ linhas internas em p faixas contíguas. Para o *worker* $w \in \{0, \dots, p - 1\}$, definimos

$$H = \left\lfloor \frac{N - 2}{p} \right\rfloor, \quad R = (N - 2) \bmod p, \quad (3)$$

$$i_{\min}(w) = 1 + wH + \min(w, R), \quad (4)$$

$$i_{\max}(w) = i_{\min}(w) + H - 1 + \mathbf{1}_{w < R}. \quad (5)$$

A propriedade de posse é persistente: a mesma *thread* conserva a mesma faixa nas fases sucessivas. O parâmetro `TILE` controla apenas a ordem de percurso dentro da faixa; ele não altera a identidade dos vizinhos.

Sob essa decomposição, o grafo de comunicação entre faixas é um caminho

$$E_{\text{adj}} = \{(w, w + 1) : 0 \leq w < p - 1\}. \quad (6)$$

Uma barreira sincroniza o conjunto completo de participantes; a dependência numérica, porém, atravessa somente as arestas de E_{adj} .

4. Modelagem causal e de custo

A estrutura de vizinhança da seção anterior permite separar correção e custo. Primeiro formalizamos as precedências realmente exigidas pelo OEH; depois usamos essa ordem causal para comparar, no caminho crítico, barreiras, semáforos e trocas de halos MPI.

4.1. Eventos, precedência e excesso de ordenação

Indexamos as fases globalmente por $k = 4m + \phi$, onde m é o passo temporal e $\phi \in \{0, 1, 2, 3\}$. Para o *worker* w , sejam S_w^k e F_w^k os eventos de início e término da fase k . A ordem de programa satisfaz

$$S_w^k \xrightarrow{\text{po}} F_w^k. \quad (7)$$

Para uma fronteira entre a e b , o consumo de um sinal local ou a conclusão de uma troca de halo cria uma precedência de sincronização

$$F_a^k \xrightarrow{\text{sw}} S_b^{k+1}, \quad (a, b) \in E_{\text{adj}}. \quad (8)$$

A relação causal usada na análise é o fecho transitivo das precedências existentes,

$$\xrightarrow{\text{hb}} = \left(\xrightarrow{\text{po}} \cup \xrightarrow{\text{sw}} \cup \xrightarrow{\text{bw}} \right)^+, \quad (9)$$

onde $\xrightarrow{\text{bw}}$ aparece apenas quando uma barreira global é introduzida. Essa notação é inspirada na ordenação causal clássica de Lamport e na semântica de sincronização das APIs utilizadas [Lamport 1978, OpenMP Architecture Review Board 2021].

Com barreira, a transição entre duas fases impõe

$$F_a^k \xrightarrow{\text{bw}} S_b^{k+1} \quad \forall a, b. \quad (10)$$

A Eq. (10) contém as arestas da Eq. (8), mas acrescenta precedências entre trabalhadores cujos subdomínios não compartilham dados. A Figura 1 ilustra essa diferença.

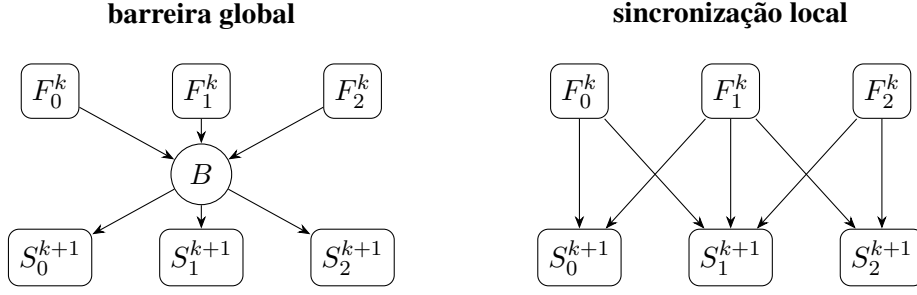


Figure 1. A barreira transforma a conclusão de todos os participantes em condição para qualquer avanço. A política local preserva apenas a ordem de programa e as precedências entre faixas adjacentes.

Proposição 1. Considere uma decomposição em faixas persistentes e um *stencil* cujo raio espacial não ultrapassa a faixa adjacente. Se, antes de iniciar a fase $k + 1$, cada *worker* aguarda a conclusão da fase k de todos os vizinhos dos quais lê dados, então a remoção da barreira global preserva as precedências de dados necessárias ao esquema.

Justificativa. As dependências internas à faixa são preservadas por ordem de programa. Toda leitura que cruza uma fronteira provém, por hipótese, de uma faixa adjacente e é precedida pela aresta de sincronização da Eq. (8). Portanto, o fecho transitivo contém todas as precedências exigidas pelo *stencil*. As arestas adicionais da Eq. (10) ordenam faixas não adjacentes e não participam de nenhuma leitura do próximo estágio. A equivalência se refere às dependências numéricas do esquema sob as hipóteses declaradas; não é uma autorização genérica para remover barreiras de qualquer aplicação. Estabelecida a correção, comparamos a seguir o custo das duas formas de ordenação.

4.2. Mecanismos de sincronização e caminho crítico

Seja C_w^k o custo de computação do *worker* w na fase k . Para a versão OpenMP com barreira, modelamos o custo coletivo por fase como

$$B_k(p) \approx \alpha \log p + \beta \Delta_g^k, \quad \Delta_g^k = \max_w C_w^k - \min_w C_w^k, \quad (11)$$

em que o termo logarítmico representa algoritmos escaláveis de barreira e o segundo termo agrega a exposição ao desequilíbrio global [Hensgen et al. 1988, Mellor-Crummey and Scott 1991]. Assim,

$$T_{\text{bar}}^k \approx \max_w C_w^k + B_k(p). \quad (12)$$

Na política local, uma faixa interior sincroniza com dois vizinhos e uma faixa de borda com apenas um. Definimos

$$S_w \approx n_{\text{viz}}(w)(c_{\text{post}} + c_{\text{wait}}) + \ell_{\text{loc}}, \quad n_{\text{viz}} \in \{1, 2\}, \quad (13)$$

onde ℓ_{loc} agrega efeitos locais de escalonamento e memória. O desequilíbrio relevante deixa de ser a amplitude de todo o time e passa a ser o maior desbalanceamento em uma aresta do grafo,

$$\delta_{\text{adj}}^k = \max_{(a,b) \in E_{\text{adj}}} |C_a^k - C_b^k|. \quad (14)$$

Para um passo com quatro fases,

$$T_{\text{bar}}^{\text{step}} \approx \sum_{k=1}^4 \max_w C_w^k + \sum_{k=1}^4 B_k(p), \quad (15)$$

$$T_{\text{sem}}^{\text{step}} \approx \sum_{k=1}^4 \max_w C_w^k + 4\bar{S} + \gamma \sum_{k=1}^4 \delta_{\text{adj}}^k, \quad (16)$$

com $\gamma \in [0, 1]$ representando a parcela do desbalanceamento adjacente que alcança o caminho crítico. Descontando a computação comum, a sincronização por semáforos é favorecida quando

$$\sum_{k=1}^4 B_k(p) > 4\bar{S} + \gamma \sum_{k=1}^4 \delta_{\text{adj}}^k. \quad (17)$$

A Eq. (17) é a hipótese de desempenho testada na Seção 7: em baixa concorrência os dois lados podem ser comparáveis; com o crescimento de p , a ordem coletiva tende a se tornar mais cara, enquanto o grau do grafo local permanece constante.

MPI representa a mesma topologia de dependências por outro mecanismo. Para uma decomposição 1D por linhas, cada rank troca halos apenas com seus vizinhos superior e inferior. Um modelo de primeira ordem para uma mensagem de tamanho M é

$$H(M) = L + \frac{M}{\mathcal{B}}, \quad (18)$$

com latência L e banda efetiva $\mathcal{B}$ [Hockney 1994]. Em cada fase, o custo crítico do MPI combina computação local e conclusão das trocas de halo. A comparação com semáforos, portanto, não é “sincronização versus comunicação”: ambos expressam dependências locais; diferem no espaço de endereçamento, no movimento explícito de dados e nos custos dos respectivos runtimes.

5. Protocolos de sincronização e implementações

As três implementações preservam o mesmo núcleo numérico, o mesmo número de fases e a mesma granularidade de *tiling*. Para isolar a política de sincronização, as duas variantes OpenMP usam o mesmo tratamento de *false sharing*, com o layout mantido fixo na comparação [Drepper 2007].

OpenMP com barreiras A versão de referência usa uma região paralela envolvendo o laço temporal e distribui o trabalho de cada fase com OpenMP. Ao fim de cada região de compartilhamento, a barreira implícita estabelece um *join* de fase. A distribuição não precisa expor a identidade dos vizinhos porque a barreira fornece uma ordem coletiva suficiente para todas as dependências [OpenMP Architecture Review Board 2021]; o layout está controlado, mas a ordem entre fases continua global.

OpenMP com semáforos direcionais A versão local estabelece posse persistente de faixas. Para cada interface entre os *workers* w e $w + 1$ existem dois canais lógicos direcionais, implementados por semáforos POSIX inicializados em zero

[IEEE and The Open Group 2018]. Após concluir uma fase, a *thread* publica um crédito para cada vizinho. Antes de iniciar qualquer fase posterior à primeira fase global, consome exatamente um crédito de cada vizinho existente. Dessa forma, cada crédito representa a conclusão da fase imediatamente anterior na interface correspondente.

O Algoritmo 1 usa um índice global de estágio para tornar explícita a correspondência um-para-um entre publicação e consumo. Não há um contador global nem um ponto de encontro coletivo.

Algorithm 1: Protocolo local por semáforos para a faixa do *worker* w

```

1 Inicialize os semáforos direcionais das interfaces com zero
2 for  $k \leftarrow 0$  to  $4T - 1$  do
3   if  $k > 0$  then
4     foreach vizinho  $q$  existente de  $w$  do
5        $\lfloor$  sem_wait no crédito publicado por  $q$ 
6   execute a fase  $k \bmod 4$  na faixa persistente de  $w$ 
7   foreach vizinho  $q$  existente de  $w$  do
8      $\lfloor$  sem_post no canal  $w \rightarrow q$ 

```

A ausência de impasse segue por indução nos estágios. O estágio $k = 0$ não possui espera antecedente. Admitindo que todas as faixas tenham concluído $k - 1$, cada uma publicou os créditos consumidos antes de k ; portanto, nenhuma espera em k depende de uma publicação que só possa ocorrer depois da própria espera. Como a computação de cada fase termina, os créditos necessários são eventualmente produzidos. O mesmo argumento mostra que não há mistura de estágios: cada fase concluída produz exatamente um crédito por direção e cada fase seguinte consome exatamente um.

O protocolo altera a política de sincronização, não o tratamento de *false sharing*; por isso, as duas curvas OpenMP compartilham a mesma base de layout. MPI fornece então o contraponto em que a localidade das dependências é realizada por processos e halos.

MPI puro Na versão MPI, as linhas internas são decompostas entre R processos. Cada rank mantém uma linha fantasma superior e uma inferior e, ao final de cada fase, troca as fronteiras necessárias com os ranks adjacentes por operações ponto a ponto não bloqueantes, concluídas antes do consumo dos halos na fase dependente. A semântica de comunicação e completude segue o padrão MPI [MPI Forum 2023]. Para uma ocupação total de W *workers*, essa variante usa $R = W$ e uma única *thread* por processo.

A Tabela 1 resume a comparação conceitual.

6. Metodologia experimental

A campanha analisada foi executada em um único nó com dois processadores AMD EPYC 9684X, totalizando 192 núcleos físicos, sem uso de SMT. O problema tem $N = 8192$ pontos por dimensão e $T = 100$ passos temporais. Foram avaliados $\text{TILE} \in \{8, 16, 32\}$ e $W \in \{1, 2, 4, 8, 16, 32, 64, 128, 160, 192\}$ *workers*.

Table 1. Variantes consideradas no artigo.

Variante	Decomposição	Coordenação entre fases
OpenMP-barreira	1 processo, W threads	<i>Join</i> coletivo de todas as threads
OpenMP-semáforos	1 processo, W threads, faixas persistentes	Créditos POSIX somente entre faixas adjacentes
MPI puro	W processos, 1 thread por processo	Troca de halos somente entre ranks adjacentes

A avaliação compara três configurações: MPI puro, OpenMP com barreiras e OpenMP com semáforos. Para isolar o efeito da política de sincronização, **as duas configurações OpenMP empregam o mesmo tratamento de *false sharing*** e mantêm o mesmo arranjo de memória. Não comparamos versões tratadas e não tratadas; o tratamento é mantido como condição experimental fixa. Dessa forma, a diferença de interesse entre as duas curvas OpenMP é o mecanismo de sincronização: ordem coletiva por barreiras ou precedências locais materializadas por semáforos.

Cada configuração possui dez execuções válidas. O tempo representativo é a mediana dessas dez observações e, para caracterizar a dispersão, empregamos o intervalo interquartil (IQR). A escolha da mediana reduz a influência de valores extremos, particularmente visíveis em algumas execuções MPI. Como o tempo de uma implementação com um *worker* difere entre famílias, a comparação principal usa **tempo absoluto**, evitando que baselines distintos produzam interpretações artificiais de *speedup*.

Para comparar diretamente as duas políticas OpenMP, definimos

$$G_{\text{sem}}(W, \text{TILE}) = \frac{T_{\text{bar}}(W, \text{TILE})}{T_{\text{sem}}(W, \text{TILE})}. \quad (19)$$

Valores $G_{\text{sem}} > 1$ indicam vantagem dos semáforos. Também reportamos a redução percentual

$$R_{\text{sem}} = 100 \left(1 - \frac{T_{\text{sem}}}{T_{\text{bar}}} \right) \quad (20)$$

e a distância relativa para MPI, $D_{\text{MPI}} = 100(T_{\text{sem}}/T_{\text{MPI}} - 1)$.

7. Resultados

A análise segue o modelo: primeiro localizamos o regime em que semáforos reduzem o tempo frente às barreiras; depois examinamos o *crossover*, a distância para MPI puro e a interação com `TILE`.

7.1. Escalabilidade em tempo absoluto

As Figuras 2, 3 e 4 apresentam os tempos medianos. O eixo vertical logarítmico permite acompanhar a transição de dezenas de segundos em baixa concorrência para frações de segundo próximo à ocupação completa do nó.

Em baixa concorrência não existe uma vantagem uniforme dos semáforos. Em `TILE= 16`, por exemplo, a variante local é 6,2% mais lenta que a barreira em duas

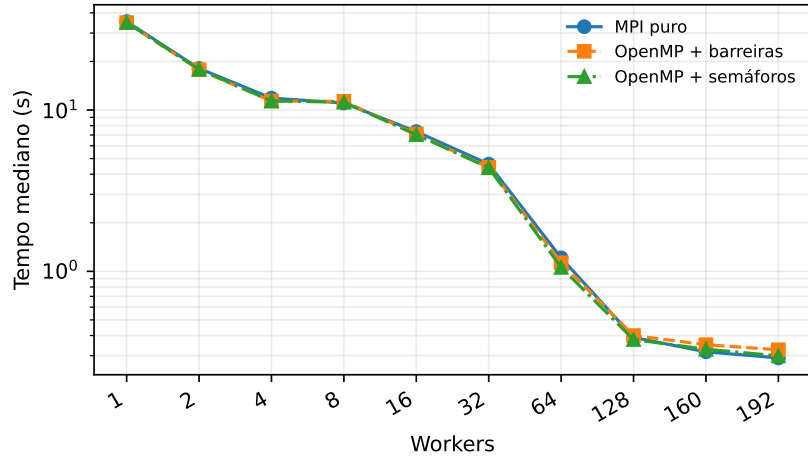


Figure 2. Tempo mediano para $N = 8192$, $T = 100$ e **TILE= 8.**

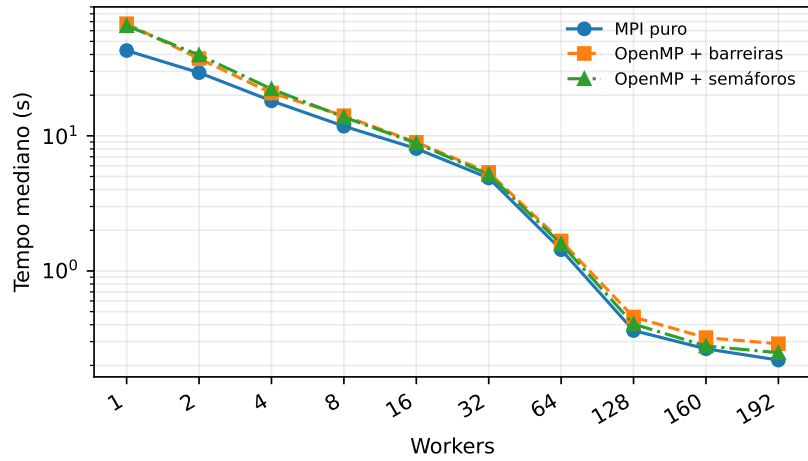


Figure 3. Tempo mediano para $N = 8192$, $T = 100$ e **TILE= 16.**

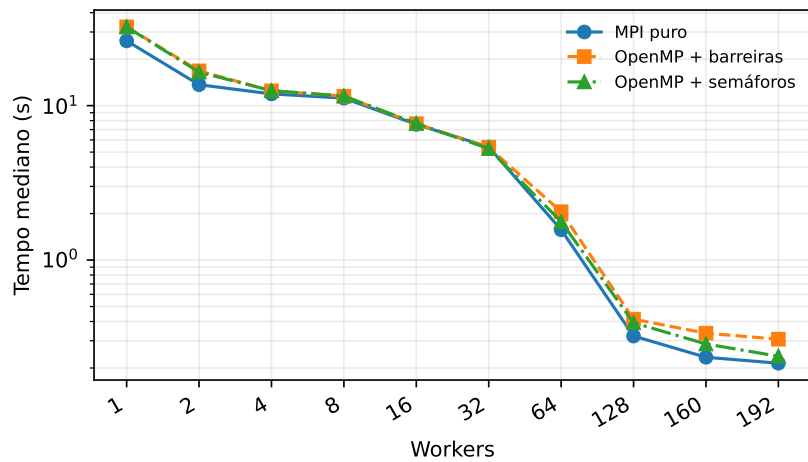


Figure 4. Tempo mediano para $N = 8192$, $T = 100$ e **TILE= 32.**

threads e 7,1% mais lenta em quatro. Em **TILE**= 32, as diferenças entre quatro e 16 threads permanecem abaixo de 0,4% em módulo. Esse comportamento é consistente com a

Eq. (17): quando $B_k(p)$ ainda é pequeno, o custo fixo de bloquear e liberar vizinhos pode neutralizar o benefício de remover o *join*.

A situação muda a partir de 32 threads. Considerando conjuntamente os três valores de `TILE` e $W \in \{32, 64, 128, 160, 192\}$, há 15 configurações, e em **todas as 15** o tempo mediano com semáforos é menor que o tempo com barreiras. O ganho cresce especialmente na região mais concorrente. Em 192 threads, os tempos $(T_{\text{bar}}, T_{\text{sem}})$ são (0,3262, 0,2994) s para `TILE`= 8, (0,2892, 0,2482) s para `TILE`= 16 e (0,3073, 0,2384) s para `TILE`= 32.

7.2. Crossover entre ordem coletiva e ordem local

A Figura 5 remove a escala absoluta do problema e mostra diretamente G_{sem} . A linha horizontal em 1 representa empate. A passagem para um regime favorável à sincronização local é clara: a razão fica acima de 1 para todos os `TILE` a partir de 32 threads e cresce para 1,090, 1,165 e 1,289 em 192 threads, respectivamente.

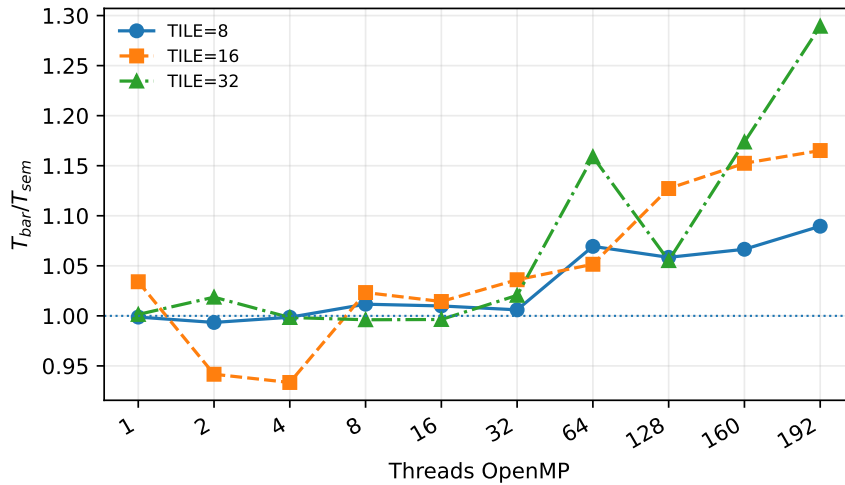


Figure 5. Razão entre o tempo com barreiras e o tempo com semáforos. Valores acima de 1 favorecem sincronização local.

Em termos de redução percentual, os semáforos economizam 8,2%, 14,2% e 22,4% em 192 threads para `TILE`= 8, 16, 32. O crescimento do ganho com a concorrência é a assinatura esperada do modelo: a barreira expõe o time completo ao participante mais lento, enquanto a política local limita a espera ao entorno espacial imediato.

O comportamento em 64 threads merece atenção. Para `TILE`= 8, 16, 32, as reduções são 6,5%, 4,9% e 13,7%, respectivamente. Portanto, o ganho não depende de atingir exatamente 192 núcleos; o regime favorável já aparece antes da ocupação total. Por outro lado, a não monotonicidade entre 64 e 128 threads, sobretudo em `TILE`= 32, mostra que a Eq. (17) é uma fronteira de primeira ordem: localidade, afinidade e interferência de memória também variam com W e afetam os termos efetivos do modelo. Estabelecido o ganho entre as políticas OpenMP, comparamos a coordenação local com a decomposição por processos.

7.3. Sincronização local: semáforos versus MPI

MPI puro é uma referência importante porque também restringe a coordenação a vizinhos. A Figura 6 compara as três variantes em 192 *workers*. O menor tempo absoluto é obtido

por MPI com `TILE= 32`: 0,2148 s. A melhor configuração dos semáforos também ocorre em `TILE= 32`, com 0,2384 s, apenas 11,0% acima do MPI e 22,4% abaixo da barreira na mesma granularidade.

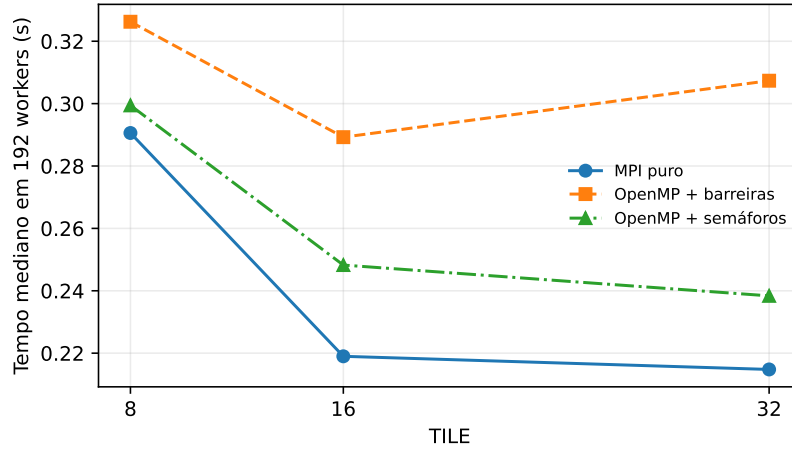


Figure 6. Sensibilidade a `TILE` com 192 *workers*.

Para `TILE= 8`, a diferença entre semáforos e MPI em 192 *workers* cai para apenas 3,0%: 0,2994 s contra 0,2906 s. Além disso, nessa granularidade, semáforos chegam a superar MPI em 32, 64 e 128 *workers*, com diferenças de 5,1%, 12,8% e 3,0% a favor do OpenMP local. Em 160 e 192 *workers*, MPI volta a ser 3,7% e 3,0% mais rápido. Esse cruzamento mostra que nenhuma das duas formas de coordenação local domina universalmente: o custo de mensagens e o custo de sincronização em memória compartilhada respondem de maneira diferente à granularidade e ao número de participantes.

A Tabela 2 apresenta também a dispersão das dez execuções. O MPI possui IQR maior nos três valores de `TILE` nesta campanha. Isso não é usado como evidência causal sobre o runtime MPI; apenas indica que, para os dados disponíveis, a mediana é uma estatística mais representativa do que a média para a comparação principal.

Table 2. Tempo mediano e IQR (s) em 192 *workers*.

<code>TILE</code>	Variante	Mediana	IQR
8	MPI puro	0,2906	0,0612
8	OpenMP-barreira	0,3262	0,0035
8	OpenMP-semáforos	0,2994	0,0050
16	MPI puro	0,2190	0,0614
16	OpenMP-barreira	0,2892	0,0027
16	OpenMP-semáforos	0,2482	0,0044
32	MPI puro	0,2148	0,0159
32	OpenMP-barreira	0,3073	0,0074
32	OpenMP-semáforos	0,2384	0,0012

7.4. Interação entre *tiling* e sincronização

Os resultados mostram ainda que a escolha de `TILE` não pode ser dissociada da política de sincronização. Em 192 threads, a barreira apresenta seu menor tempo em `TILE= 16`

(0,2892 s), enquanto os semáforos melhoram de 0,2994 s em `TILE= 8` para 0,2384 s em `TILE= 32`. Em outras palavras, a política local torna a granularidade mais grossa particularmente atraente. Isso é coerente com a literatura de *stencils*: *tiling* altera localidade e quantidade de trabalho útil entre eventos de coordenação [Datta et al. 2008]. O dado importante aqui é que esse efeito não é apenas uma otimização de cache isolada; ele modifica a relação entre os termos dos dois lados da Eq. (17).

8. Discussão e limitações

A evidência experimental é consistente com dois regimes de sincronização. Com poucos participantes, a barreira ainda é barata e os semáforos produzem empate ou pequenas perdas. A partir de aproximadamente 32 *threads* nesta campanha, a ordem coletiva passa a custar mais que a coordenação local e os semáforos vencem sistematicamente. Isso não implica superioridade universal: a Eq. (17) prevê que, quando p é pequeno ou o bloqueio local é comparável ao custo coletivo, a vantagem pode desaparecer. O resultado geral é que a topologia de sincronização deve acompanhar a topologia das dependências, evitando ordem global que não é consumida pelo método.

A campanha usa uma única malha ($N = 8192$), três valores de `TILE`, um nó e um método numérico. Como os dados não incluem contadores de hardware, energia ou decomposições híbridas, não inferimos causas microarquiteturais nesses regimes. O modelo de custo é de primeira ordem e não reproduz cache, NUMA ou escalonamento ciclo a ciclo; além disso, a comparação MPI/OpenMP usa tempos absolutos, pois os baselines com um *worker* são distintos.

9. Conclusão

Neste trabalho, mostramos que, para *stencils* de raio um com decomposição em faixas, as dependências numéricas são locais e não exigem a ordenação global imposta por barreiras. Com base nessa observação, propusemos um protocolo de sincronização orientada por dependências, baseado em créditos direcionais e semáforos POSIX, que materializa apenas as precedências entre faixas vizinhas.

No estudo de caso do Odd–Even Hopscotch para a equação do calor 2D, com $N = 8192$, $T = 100$ e `TILE` $\in$ 8, 16, 32, os semáforos superaram as barreiras nas 15 configurações avaliadas entre 32 e 192 *threads*. Em 192 *threads*, as reduções do tempo mediano foram de 8,2%, 14,2% e 22,4% para `TILE` = 8, 16 e 32, respectivamente. O MPI puro permaneceu como referência de menor tempo absoluto, mas a diferença para os semáforos foi de apenas 3,0% em `TILE= 8` e 11,0% em `TILE= 32`. Os resultados são consistentes com o modelo de custo proposto e indicam que, em alta concorrência, a redução da espera coletiva pode compensar o custo adicional da sincronização local.

Pretendemos estender para outros estudos de caso que possam se beneficiar da sincronização entre vizinhos e da ordenação causada por barreiras, em que um stencil de raio um requer precedência apenas entre interfaces. Por fim, pretendemos avaliar a sincronização orientada por dependências para *stencils* em arquiteturas com números ainda maiores de núcleos.

References

- [Cabral et al. 2020] Cabral, F. L., Oliveira, S. L. G. d., Osthoff, C., Costa, G. P., Brandão, D. N., and Kischinhevsky, M. (2020). An evaluation of mpi and openmp paradigms

- in finite-difference explicit methods for pdes on shared-memory multi- and manycore systems. *Concurrency and Computation: Practice and Experience*, 32(20):e5642.
- [Datta et al. 2008] Datta, K., Murphy, M., Volkov, V., Williams, S., Carter, J., Oliker, L., Patterson, D. A., Shalf, J., and Yelick, K. A. (2008). Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, pages 1–12.
- [Drepper 2007] Drepper, U. (2007). What every programmer should know about memory. Technical report, Red Hat.
- [Gourlay 1970] Gourlay, A. R. (1970). Hopscotch: A fast second-order partial differential equation solver. *IMA Journal of Applied Mathematics*, 6(4):375–390.
- [Gropp et al. 2014] Gropp, W., Lusk, E., and Skjellum, A. (2014). *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. MIT Press, Cambridge, MA, 3 edition.
- [Hager and Wellein 2010] Hager, G. and Wellein, G. (2010). *Introduction to High Performance Computing for Scientists and Engineers*. CRC Press, Boca Raton, FL.
- [Hensgen et al. 1988] Hensgen, D., Finkel, R., and Manber, U. (1988). Two algorithms for barrier synchronization. *International Journal of Parallel Programming*, 17(1):1–17.
- [Hockney 1994] Hockney, R. W. (1994). The communication challenge for mpp: Intel paragon and meiko cs-2. *Parallel Computing*, 20(3):389–398.
- [IEEE and The Open Group 2018] IEEE and The Open Group (2018). *IEEE Standard for Information Technology—Portable Operating System Interface (POSIX) Base Specifications, Issue 7*. IEEE and The Open Group.
- [Lamport 1978] Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565.
- [Mellor-Crummey and Scott 1991] Mellor-Crummey, J. M. and Scott, M. L. (1991). Algorithms for scalable synchronization on shared-memory multiprocessors. *ACM Transactions on Computer Systems*, 9(1):21–65.
- [MPI Forum 2023] MPI Forum (2023). MPI: A message-passing interface standard, version 4.1. Technical report, MPI Forum.
- [OpenMP Architecture Review Board 2021] OpenMP Architecture Review Board (2021). *OpenMP Application Programming Interface, Version 5.2*. OpenMP Architecture Review Board.
- [Rabenseifner et al. 2009] Rabenseifner, R., Hager, G., and Jost, G. (2009). Hybrid mpi/openmp parallel programming on clusters of multi-core smp nodes. *Proceedings of the 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, pages 427–436.
- [ten Thije Boonkkamp and Verwer 1987] ten Thije Boonkkamp, J. H. M. and Verwer, J. G. (1987). On the odd-even hopscotch scheme for the numerical integration of time-dependent partial differential equations. *Applied Numerical Mathematics*, 3(1):183–193.