3
ParTopS: suporte topológico compacto em paralelo para representação de fraturas

Neste capítulo, propõe-se um suporte topológico para a representação de fraturas baseada em modelos de zona coesiva em um ambiente paralelo. Inicialmente, é desenvolvida uma representação topológica distribuída de malhas de elementos finitos para simulações de fraturas e fragmentação dinâmicas. Em seguida, com base na representação de malha distribuída, é proposto um algoritmo paralelo para a inserção dinâmica de elementos coesivos. O suporte topológico em paralelo é chamado ParTopS, e implementado como uma extensão da estrutura topológica sequencial TopS (Celes, Paulino & Espinha; 2005a, b). Seguindo a filosofia de TopS, a malha distribuída é mantida de maneira compacta, mas com o conjunto de funcionalidades necessárias à representação eficiente de fraturas.

3.1.
Representação de malha distribuída

A malha distribuída consiste na união das partições disjuntas do conjunto de entidades topológicas que compõem a malha original de elementos finitos (Figura 12). Cada partição de malha corresponde a uma unidade de computação sequencial, executada por um processador distinto.

Durante o particionamento inicial da malha, cada elemento ou nó é atribuído a uma única partição (Figura 12). Porém, nós das fronteiras de partições podem ser compartilhados por elementos de partições diferentes. Para se manter a consistência topológica dos elementos de cada partição, são criadas cópias dos nós correspondentes em cada partição que não possui os nós originais. Em ParTopS, essas cópias são fornecidas pela *camada de comunicação*, apresentada na Seção 3.1.1, que é adicionada às fronteiras de cada partição. Entidades implícitas (vértice, aresta, face e faceta) são atribuídas às mesmas partições que os elementos aos quais se encontram ancoradas (Seção 2.6.1).

Na representação de malha distribuída proposta, a malha local de cada partição é representada pela estrutura de dados topológica TopS, estendida com uma infraestrutura adicional para a comunicação entre partições. As extensões de TopS que formam a representação distribuída de ParTopS são descritas nas subseções seguintes.

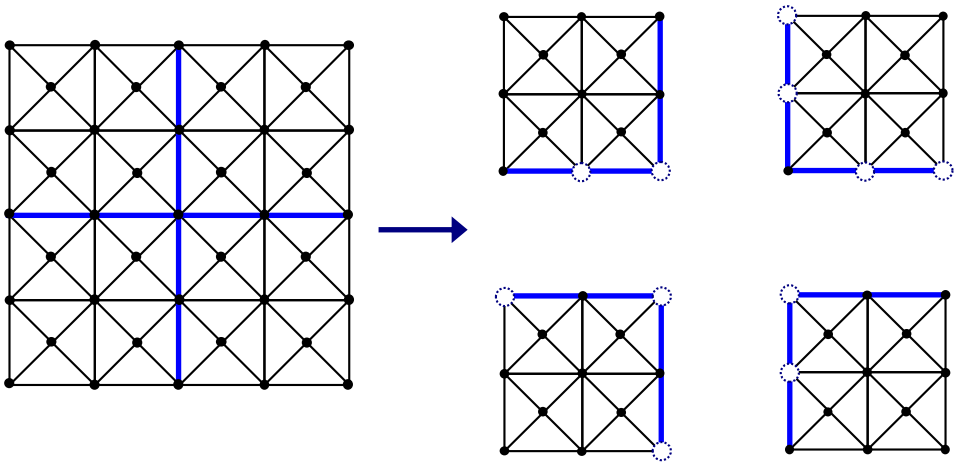

Figura 12 - (a) Malha de elementos finitos original; (b) partições da malha de elementos finitos.

3.1.1. Camada de comunicação

Durante a análise de elementos finitos, computações realizadas em elementos e nós precisam acessar dados armazenados em entidades adjacentes (ex.: a computação de tensões nodais requer contribuições dos elementos adjacentes aos nós). Todavia, elementos e nós localizados nas fronteiras de uma partição podem ser adjacentes a entidades representadas em outras partições. Assim, torna-se necessário que a partição se comunique com as vizinhas de modo a obter os dados necessários (Figura 13a).

Em relação à representação de elementos coesivos, podem ocorrer inconsistências topológicas quando esses elementos são representados em fronteiras compartilhadas por partições de malha diferentes. Por definição, elementos coesivos só existem em interfaces entre dois elementos volumétricos adjacentes, porém os elementos volumétricos podem ser representados por partições diferentes. Segundo a classificação de Paulino et al. (2008), a

topologia de um elemento coesivo também é determinada a partir de informações topológicas do conjunto de elementos adjacentes aos vértices, arestas e facetas do elemento. Na inserção de elementos coesivos, a decisão se um nó do elemento deve ser duplicado, e como será duplicado, por exemplo, depende do conjunto de elementos adjacentes. Entretanto, nas fronteiras compartilhadas de uma partição, o conjunto local de elementos adjacentes não é completo.

A Figura 13b ilustra a interface de um elemento volumétrico localizado na fronteira de uma partição, onde se pretende inserir um novo elemento coesivo. A inserção do elemento requer operações topológicas baseadas nas entidades adjacentes, como buscas locais ao redor de vértices, arestas e facetas. Para que o elemento coesivo possa ser criado, é necessário que a topologia da região de malha ao redor da interface esteja completa. Isso também é necessário para que a representação do elemento na malha distribuída esteja consistente com sua definição.

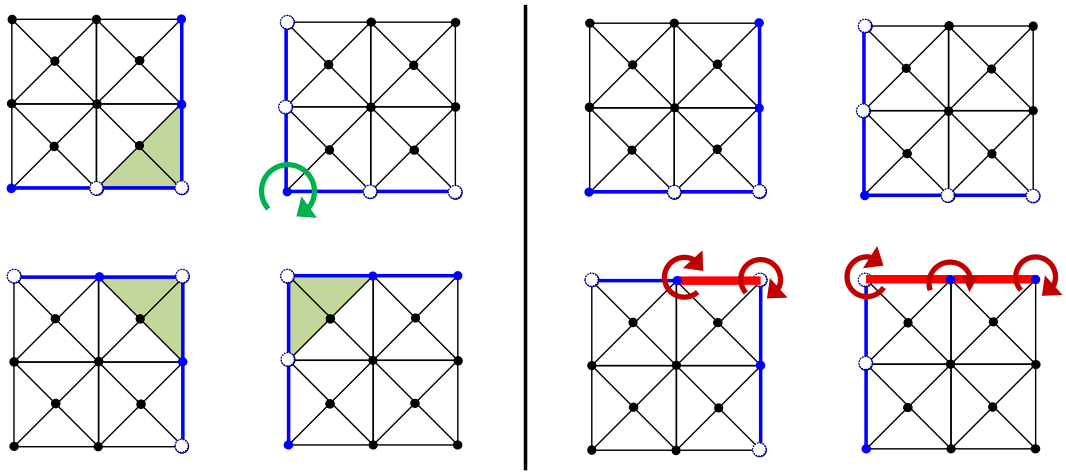

Figura 13 - (a) Computação ao redor de um nó requer dados de elementos adjacentes representados em outras partições. (b) A topologia da malha ao redor de um elemento coesivo deve ser completa para a determinação de duplicações nodais necessárias. Isso requer o acesso a elementos volumétricos adjacentes que podem estar representados em partições diferentes.

Conforme discutido na Seção 2.1, uma maneira comum de se fornecer a uma partição o acesso consistente a dados de entidades topológicas de

partições vizinhas consiste na criação de cópias das entidades remotas (representadas pelas partições vizinhas) na partição local corrente. A fim de se reduzir a quantidade de comunicação necessária para o acesso à vizinhança completa de uma entidade, uma *camada de comunicação* é criada ao redor das fronteiras da partição (Figura 14). A camada de comunicação é composta por cópias de todos os elementos remotos adjacentes aos nós das fronteiras da partição corrente, mais os nós desses elementos. Ela é criada logo após o particionamento inicial da malha original e mantida durante o tempo de vida de cada partição.

Em ParTopS, uma entidade topológica da malha original é chamada *entidade real*, e a partição de malha à qual ela está atribuída é definida como a *partição proprietária* da entidade. Embora, devido à camada de comunicação, diversas partições possam ter uma cópia de uma mesma entidade real, apenas uma é a proprietária da entidade. Nessa partição, a entidade real é representada por uma *entidade local* correspondente. Em outras, ela é representada pelas camadas de comunicação correspondentes das partições.

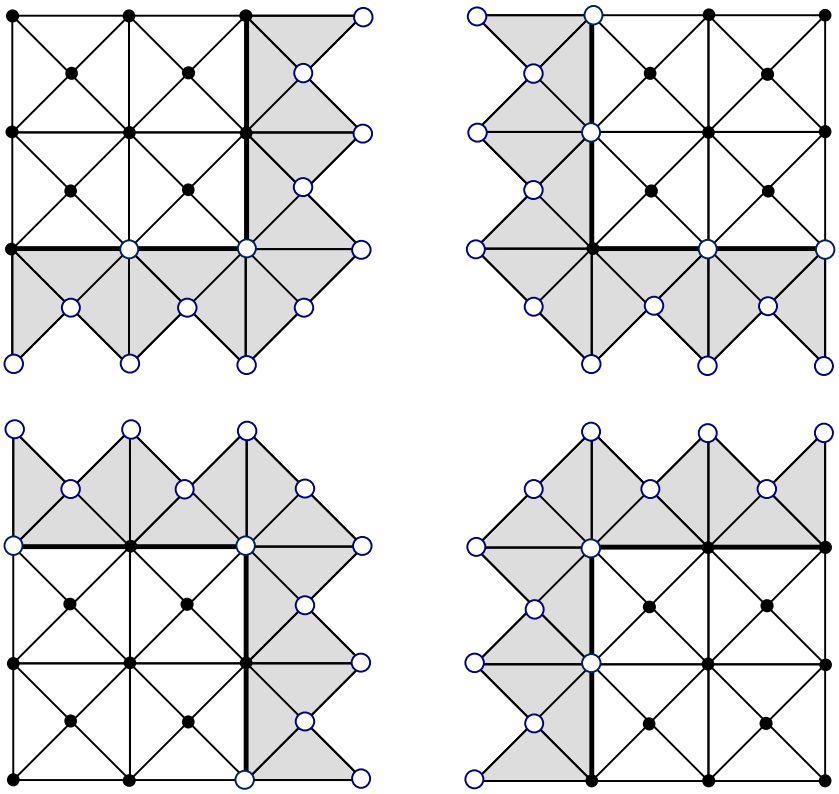

Figura 14 - Camada de comunicação; cópias de entidades remotas são adicionadas ao redor dos nós das fronteiras de cada partição.

Com a introdução de cópias de entidades remotas, a topologia ao redor das fronteiras originais das partições torna-se completa. Porém, a topologia das novas fronteiras, definidas agora pela camada de comunicação, permanece incompleta. Para expressar essa distinção de forma consistente, diferentemente de abordagens anteriores, propõe-se que as entidades topológicas da camada de comunicação sejam classificadas em dois tipos: *proxy* e *ghost*. A definição desses tipos é baseada no conceito de *vizinhança direta* de entidades, apresentado a seguir.

Define-se a *vizinhança direta* de uma entidade topológica como o conjunto de entidades, de dimensão diferente da entidade em questão, que são diretamente adjacentes a ela na malha de elementos finitos original, considerando-se uma representação de malha completa (contendo todas as entidades topológicas representáveis). Sem perda de generalidade, isso é ilustrado na Figura 15, para um nó (Figura 15a) e um elemento (Figura 15b) de uma malha bidimensional. A vizinhança direta do nó corresponde ao conjunto de elementos e outras entidades (arestas, faces e facetas) que incidem diretamente sobre ele. Por outro lado, a vizinhança direta do elemento é composta pelos nós e outras entidades (vértices, arestas e facetas) adjacentes ao elemento. A vizinhança direta de uma aresta é mostrada na Figura 15c; ela consiste no conjunto de nós, elementos e outras entidades (vértices e faces) adjacentes à aresta.

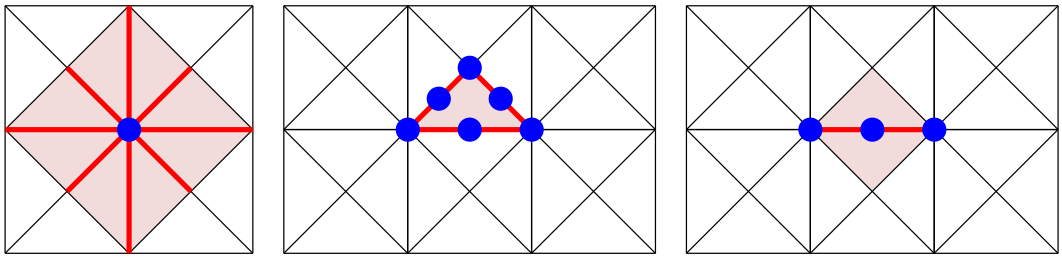

Figura 15 - (a) Vizinhança direta de um nó; (b) vizinhança direta de um elemento; (c) vizinhança direta de uma aresta.

A partir da vizinhança direta de entidades, podemos, então, classificar as entidades da camada de comunicação. Uma entidade da camada de comunicação é do tipo *proxy* quando a sua vizinhança direta é completamente representada, em relação à malha original, na partição corrente. Caso contrário, a entidade é do tipo *ghost*. Na Figura 16a, são destacados nós dos tipos *proxy* e

ghost da camada de comunicação de uma partição. Elementos são sempre do tipo *proxy* (Figura 16b), uma vez que todos os seus nós e entidades implícitas adjacentes são representados na partição.

Entidades implícitas são definidas com base em elementos adjacentes. Dessa forma, para se classificar uma entidade implícita, primeiramente determina-se o tipo do elemento adjacente ao qual ela está ancorada na partição corrente. Se o elemento é *local*, então a entidade também será uma entidade *local* na partição. Caso contrário (o elemento é *proxy*), ela será parte da camada de comunicação. Nesse caso, a vizinhança direta da entidade é utilizada para classificá-la como *proxy* ou *ghost*, de maneira similar a nós e elementos. Na Figura 16c, são destacadas as arestas *proxies* e *ghosts* da camada de comunicação de uma partição. As âncoras de entidades (Seção 2.6.1) das fronteiras compartilhadas originais da partição são mostradas (marca 'x') próximas às arestas locais ou *proxies* correspondentes.

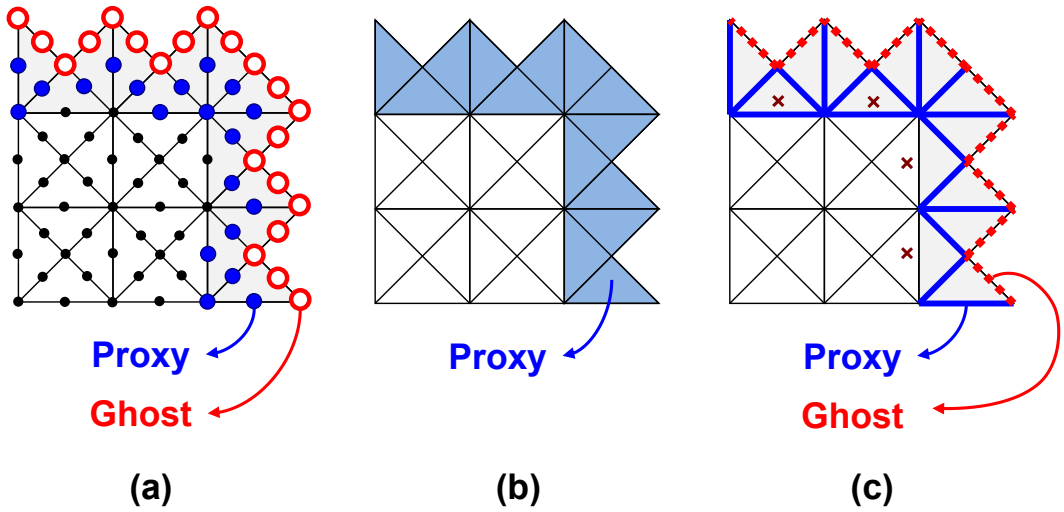

Figura 16 - (a) Nós do tipo *proxy* e *ghost* são enfatizados; (b) todos os elementos da camada de comunicação são do tipo *proxy*; (c) arestas dos tipos *proxy* e *ghost* são destacadas. As âncoras de arestas das fronteiras compartilhadas originais (marca 'x') indicam a classificação das arestas correspondentes.

Todo elemento e nó da camada de comunicação possui uma referência para a entidade real correspondente, representada por uma entidade local na partição proprietária da entidade (Figura 17). A referência é definida pelo par ordenado: *(owner_partition, owner_handle)*, onde *owner_partition* é o identificador da partição proprietária, e *owner_handle* é o identificador local da

entidade naquela partição. Note que o par *(owner_partition, owner_handle)* pode ser usado para identificar univocamente qualquer entidade real na malha distribuída, sem a necessidade da atribuição explícita de identificadores globais.

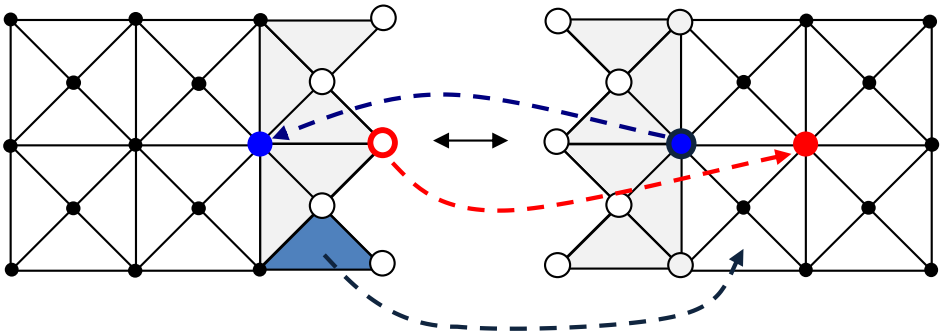

Figura 17 - Referências de elemento *proxy* e nós dos tipos *proxy* e *ghost* para as entidades reais correspondentes.

Ao contrário da abordagem convencional, as entidades da camada de comunicação são consideradas *editáveis*. Assim, podem ter topologia e atributos de simulação modificados pela aplicação cliente, da mesma forma que entidades locais. Uma vez que a vizinhança direta de entidades locais e *proxies* é completa, operações topológicas e computações que dependem de entidades adjacentes a uma entidade local ou *proxy* podem ser independentemente realizadas por cada partição. Isso torna a edição de *proxies* conveniente para a redução da comunicação entre partições, através do uso de *operações simétricas*, discutidas na Seção 3.2.1, para a inserção de elementos coesivos, e no Capítulo 4, no contexto da paralelização de simulações de fraturas.

Em ParTopS, a definição de entidades do tipo *ghost* é empregada para definir as fronteiras da camada de comunicação. A distinção entre *proxies* e *ghosts* é fundamental para a inserção paralela de elementos coesivos proposta na Seção 3.2, que a utiliza para resolver ambiguidades de elementos coesivos localizados próximos às fronteiras das partições. Em relação à representação de malha distribuída, entidades do tipo *ghost* possuem duas funções importantes: garantir a consistência topológica local de cada partição, e prover atributos de simulação necessários a computações que dependem de entidades adjacentes. Com isso, a topologia de entidades *ghosts* deve ser consistente com relação à malha sequencial de uma partição. Porém, isso não é requerido em relação à malha global original.

Para garantir a consistência topológica local de uma partição, as relações de adjacência de entidades *ghosts* são mapeadas para as entidades adjacentes

presentes na partição. O mesmo ocorre para as âncoras (Seção 2.6.1) das entidades implícitas correspondentes, relativas a entidades remotas não representadas pela partição. Isso é ilustrado na Figura 18, para o caso de nós. Em uma malha sequencial de TopS, todo nó mantém uma referência para um dos elementos adjacentes (relação nó-a-elemento). Porém, na malha distribuída da Figura 18, o nó *ghost* destacado possuiria uma referência para um elemento remoto que não é representado pela partição do nó. Nesse caso, a referência do nó é, então, atribuída a um dos elementos *proxies* adjacentes a ele. Essa referência, todavia, difere da correspondente na malha original, representada pelo nó local da outra partição. A referência de um nó *proxy* para o elemento correspondente, por outro lado, é a mesma em ambas as partições. Essa correspondência garante que entidades *proxies* são topologicamente consistentes em relação à malha global.

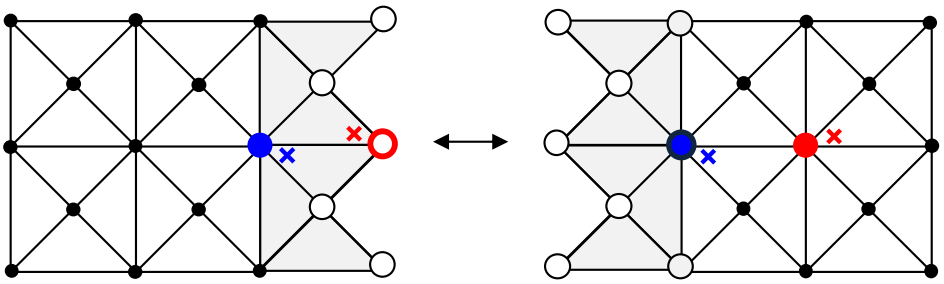

Figura 18 - Consistência topológica local de um nó *proxy* e outro *ghost*. Embora a malha local de cada partição seja consistente, a referência do nó *ghost* ao elemento adjacente é diferente em cada partição. Por outro lado, a referência do nó *proxy* é idêntica.

Propriedades adicionais da camada de comunicação são descritas a seguir:

- *Todos os elementos adjacentes a um nó ghost são do tipo proxy.* De forma equivalente, nós do tipo *ghost* não são adjacentes a elementos locais. Considerando-se que a camada de comunicação é criada ao redor de todos os nós das fronteiras originais de uma partição, então a vizinhança direta desses nós é completa. Assim, se um nó *ghost* fosse adjacente a um elemento local, a vizinhança dele seria completa, o que contraria a definição desse tipo de nó.
- *Uma entidade implícita da camada de comunicação é do tipo ghost se e somente se todos os nós adjacentes a ela também são ghosts*.

Se uma faceta *ghost* fosse adjacente a um nó *proxy* ou local, então todos os elementos adjacentes a ela seriam representados. Como consequência, as outras entidades implícitas também seriam representadas localmente, e a vizinhança direta da entidade seria completa, contrariando a definição de *ghost*. O mesmo ocorre para faces e arestas. No caso de um vértice, ele é determinado pelo tipo do nó correspondente.

3.1.2. Construção da camada de comunicação

Quando uma partição de malha é criada, um subconjunto disjunto dos elementos e nós correspondentes da malha original é inicialmente atribuído a ela (Figura 19a). Esses elementos e nós são representados como entidades locais da partição. Porém, elementos são definidos por um conjunto ordenado de nós, alguns dos quais podem ter sido atribuídos a partições diferentes. Assim, *proxies* são adicionados à partição para representar os nós remotos correspondentes (Figura 19b). Em seguida, elementos *proxies*, juntamente com os respectivos nós, são criados ao redor de todos os nós localizados nas fronteiras da partição e que possuem elementos adjacentes remotos (Figura 19c). Finalmente, os nós de elementos *proxies* que não possuem a vizinhança direta completa representada pela partição são classificados como *ghosts* (Figura 19d).

Considerando-se que elementos são representados de forma consistente entre as partições da malha, entidades implícitas também serão se um critério uniforme for usado para determinar os elementos aos quais estão ancoradas. O critério adotado aqui consiste no elemento adjacente com a menor referência à entidade real correspondente. Assim, o par ordenado *(owner_partition, owner_handle)* – Seção 3.1.1 – é comparado em ordem lexicográfica, primeiramente *owner_partition*, e em seguida *owner_handle*. Com isso, as âncoras de entidades locais e *proxies* implícitas são determinadas de maneira simétrica sem a necessidade de comunicação entre partições.

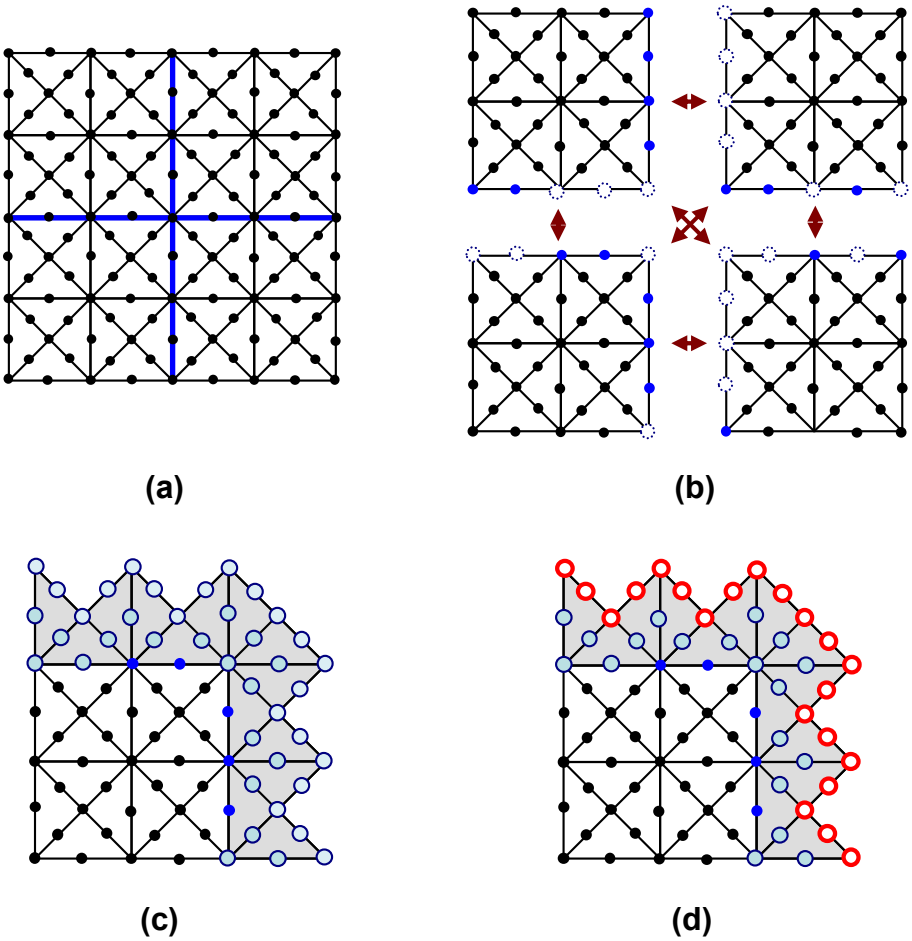

Figura 19 - Criação da camada de comunicação. (a) malha original; (b) malha particionada; (c) partição de malha com elementos e nós *proxies* iniciais; (d) partição de malha final; nós *proxies* que não possuem vizinhança direta completa são convertidos em *ghosts*.

3.1.3. Vizinhança de uma partição

Embora a malha distribuída possa ser composta por um grande número de partições da malha original, em geral, cada partição precisa se comunicar apenas com um pequeno número de outras localizadas ao redor dela. Dessa forma, para que a comunicação entre partições possa ser feita de maneira eficiente, é necessário que cada partição conheça o seu respectivo conjunto de partições vizinhas. Isso determina o número de mensagens que a partição espera enviar ou receber de outras durante uma rodada de comunicação.

Em ParTopS, duas partições são consideradas *vizinhas* se e somente se uma delas possui uma entidade *proxy* correspondente a uma entidade local da outra. Essa definição é conveniente para a representação de malhas de fraturas

dinâmicas, pois a inserção de novos elementos coesivos não altera a vizinhança de uma partição (Seção 3.2), uma vez que cada elemento coesivo criado (ou nó duplicado devido à inserção do elemento) é atribuído à partição proprietária de um dos elementos adjacentes à nova entidade. Essa partição é a própria partição local ou outra que faz parte do conjunto de vizinhas a ela. Com isso, a vizinhança de uma partição não se modifica, e pode ser determinada apenas uma única vez, durante a criação da malha distribuída.

Se entidades do tipo *ghost* fossem consideradas na determinação de vizinhança de partições, as listas de vizinhança precisariam ser atualizadas dinamicamente com a inserção de novos elementos coesivos; isso é ilustrado na Figura 20. Quando um nó *ghost* é duplicado devido à inserção de elementos coesivos, o novo nó criado deve ser atribuído a uma partição proprietária. Porém, essa partição pode não fazer parte da vizinhança da partição que contém o nó *ghost*, e assim as listas de vizinhança de ambas as partições devem ser atualizadas para incluir uma a outra. Na Figura 20, a duplicação do nó destacado faz com que as partições P1 e P3 se tornem vizinhas. A fim de que as vinhanças das partições não precisassem ser atualizadas, todas as partições contendo elementos adjacentes ao nó duplicado deveriam ser previamente consideradas vizinhas entre si. Por esse motivo, optou-se pela definição de vizinhança baseada em entidades *proxies*, para o propósito de simulações de propagação de fraturas e fragmentação.

Considerando-se que entidades *ghosts* não influenciam a vizinhança de uma partição, a partição proprietária de uma entidade *ghost* pode não fazer parte do conjunto de vizinhas da partição corrente. Assim, a referência da entidade *ghost* para a entidade real correspondente não pode ser usada para atualizar a entidade *ghost*. Porém, observa-se que uma entidade *ghost* pode ser atualizada a partir de um dos elementos adjacentes a ela, pois toda entidade *ghost* é adjacente a um elemento *proxy*, cuja partição proprietária, por definição, faz parte da vizinhança da partição corrente. Da mesma forma, na partição proprietária do elemento *proxy*, a entidade *ghost* será representada por uma entidade local ou *proxy*, adjacente ao elemento local correspondente. Por hipótese, a topologia de um elemento é consistente em todas as partições em que está presente. Então, a entidade *ghost* possui a mesma identificação em relação à ordenação topológica local do elemento em ambas as partições. Logo, requisições de dados para a atualização da entidade *ghost* são enviadas à partição proprietária de um dos elementos *proxies* adjacentes a ele, utilizando-se a tupla: *(owner_partition, owner_element_handle, local_id)*, onde *owner_partition*

é a partição proprietária do elemento *proxy*, *owner_element_handle* é o identificador do elemento naquela partição, e *local_id* é o índice da entidade *ghost* na ordem de incidência local fixa do elemento. A atualização de um nó *ghost* é ilustrada na Figura 21.

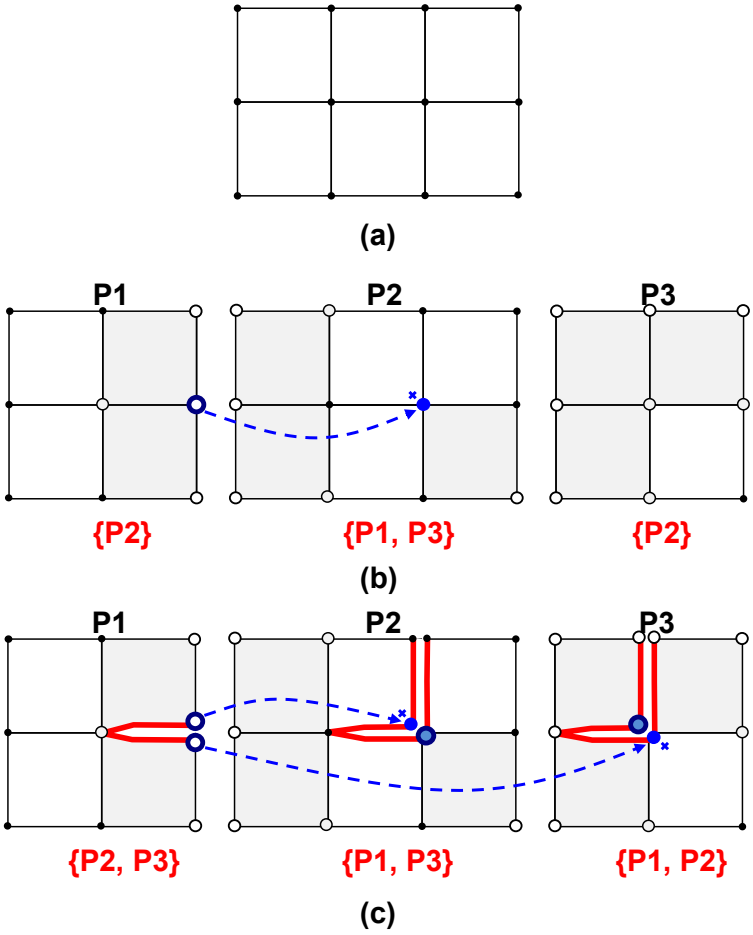

Figura 20 - Um caso em que a duplicação de um nó afeta as vizinhanças de partições, quando nós do tipo *ghost* são considerados na definição de vizinhança. (a) Malha original; (b) Três partições (*P1, P2, P3*) da malha original. A referência do nó *ghost* em destaque em *P1* aponta para o nó correspondente em *P2*; (c) Com a duplicação do nó, devido à inserção de elementos coesivos, o nó criado passa a apontar para o nó correspondente em *P3*. Dessa forma, as vizinhanças de *P1* e *P3* devem ser alteradas para incluir uma a outra.

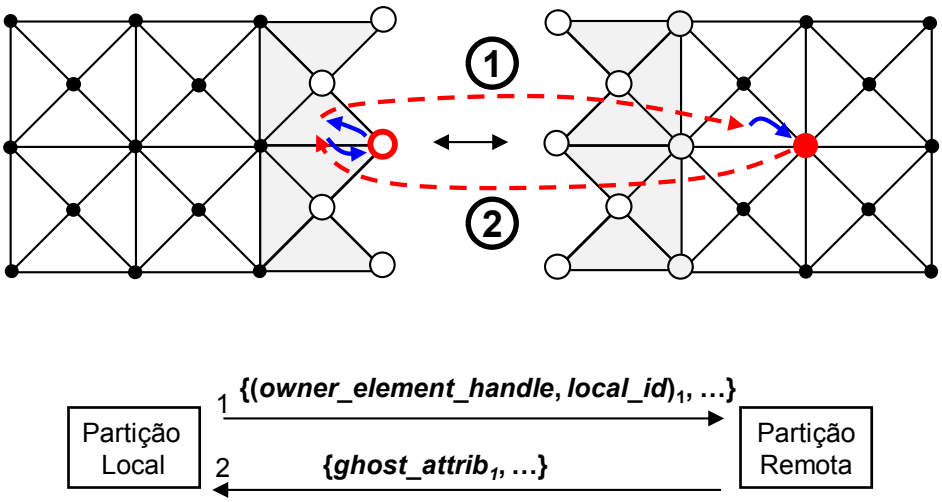

Figura 21 - Requisição para atualização de um nó *ghost* (1) é enviada, a partir de elemento *proxy* adjacente a ele, à partição proprietária do elemento. Naquela partição, o nó correspondente é do tipo local ou *proxy*. Os atributos do nó (*ghost_attrib*) são então acessados e retornados à partição requisitante (2), e o nó *ghost* é atualizado com os novos valores.

3.1.4. Representação distribuída de atributos

Atributos da simulação numérica associados a nós e elementos da malha de elementos finitos são representados por *conjuntos de atributos densos*, fornecidos por TopS (Seção 2.6). Algumas extensões são adicionadas por ParTopS para permitir a representação distribuída e a sincronização de atributos entre partições de malha vizinhas.

Um conjunto distribuído de atributos é criado através de uma função paralela coletiva, executada concorrentemente por todas as partições de malha. A criação do conjunto, entretanto, não requer comunicação entre partições. Em cada partição, a função paralela delega a criação do conjunto para uma função sequencial correspondente, que constroi conjuntos locais relacionados ao tipo de entidade desejado. A interface da função paralela é baseada apenas nos tipos de nós e elementos locais (ex. *T3*, *T6*, *Tet4*, *Tet10*, *Hex8*, *Hex20*, etc.) referentes à malha sequencial original. Porém, em cada partição, são criados ao mesmo tempo até três conjuntos de atributos, um para cada representação de entidade da malha distribuída: *local*, *proxy* e *ghost*. Assim, um conjunto distribuído de atributos de um nó será representado em cada partição por três conjuntos de atributos (*local proxy* e *ghost*). Da mesma forma, um elemento *Tet4*, por

exemplo, será representado em cada partição por dois conjuntos locais à partição: *Tet4 - local* e *Tet4 - proxy.* Uma vez que todos os conjuntos distribuídos de atributos são criados pela mesma função paralela, um contador local é usado por cada partição como identificador global dos conjuntos criados. Os conjuntos locais de cada partição podem ser acessados independentemente a partir do identificador único do conjunto distribuído.

A sincronização de um conjunto distribuído de atributos é realizada através de uma função paralela que depende do tipo de entidade à qual o conjunto está associado e do identificador global do conjunto. Para que os dados possam ser transmitidos através da rede de comunicação, uma *função de chamada de retorno* (*callback function*) é registrada pela aplicação junto ao conjunto de atributos. A função é responsável por *empacotar* os dados em grupos de *bytes* para transmissão, e *desempacotá-los* ao serem recebidos por uma partição.

Atributos de entidades *proxies* e *ghosts* são atualizados separadamente. Para cada entidade (elemento ou nó) *proxy*, a partição corrente determina a partição vizinha proprietária da entidade (*owner_partition*) e o identificador da entidade naquela partição (*owner_handle*). Em seguida, envia para cada partição vizinha correspondente uma mensagem contendo os identificadores das entidades remotas (*owner_handle*). A partição vizinha, então, acessa diretamente cada entidade, a partir do identificador recebido, e obtém os atributos do conjunto local a ela. Uma mensagem contendo os dados dos atributos é retornada à partição corrente, que atualiza as entidades *proxies*.

Nós do tipo *ghost* são sincronizados a partir de um dos elementos volumétricos *proxies* adjacentes. Para cada nó *ghost*, a partição corrente escolhe um elemento *proxy* adjacente e determina o índice do nó (*local_id*) na ordem de incidência local fixa do elemento (Seção 2.6), a partição proprietária do elemento (*owner_partition*), e o identificador do elemento naquela partição (*owner_handle*). Em seguida, envia para cada partição vizinha correspondente uma mensagem contendo pares de identificadores de elementos e índices de nós nos elementos (*owner_handle, local_id*). A partição vizinha acessa o respectivo elemento local e então o nó relativo ao índice recebido, e retorna uma mensagem com os atributos requisitados à partição corrente, que atualiza os nós *ghosts*. Uma vez que o nó da partição vizinha pode ser do tipo local ou *proxy*, é necessário que os atributos de nós *proxies* encontrem-se atualizados.

3.2. Inserção dinâmica de elementos coesivos em paralelo

Nesta seção, é proposto um algoritmo baseado em topologia para a inserção dinâmica de elementos coesivos em paralelo. Diferentemente de outras abordagens, elementos coesivos são *explicitamente* representados e podem ter atributos associados da mesma forma que outros elementos regulares da malha de elementos finitos; os elementos coesivos são criados sob demanda, e inseridos apenas onde e quando necessários à simulação, conforme requerido por modelos coesivos extrínsecos. O algoritmo paralelo pode ser aplicado de maneira uniforme tanto em malhas 2D quanto 3D, de diferentes tipos de elementos finitos.

No algoritmo proposto, a representação de malha distribuída de ParTopS, descrita na Seção 3.1, é combinada com a classificação topológica sistemática de facetas introduzida por Paulino et al. (2008). São exploradas operações topológicas que produzem resultados simétricos em todas as partições de malha para uma mesma entidade topológica. Isso permite eliminar a necessidade de acesso exclusivo (*locks*) a entidades por uma única partição, e assim reduzir a comunicação entre partições para a atualização da topologia da malha.

3.2.1. Operações topológicas simétricas

Uma dificuldade fundamental para a inserção adaptativa de elementos coesivos em paralelo está relacionada à duplicação de nós localizados nas fronteiras de uma partição (ou da camada de comunicação), que ocorre durante a criação de novos elementos coesivos. Para determinar se um nó deve ser duplicado, e assim manter a topologia da malha consistente, uma partição pode depender de modificações ocorridas em outra, mas que não são conhecidas por ela. Isso resulta em dependências cíclicas entre partições, as quais devem ser resolvidas de forma apropriada.

Para garantir a consistência topológica entre as partições, propõe-se uma abordagem baseada em *operações topológicas simétricas*. Uma *operação topológica simétrica* é definida, aqui, como uma operação topológica que é executada sobre um conjunto de entidades compartilhadas por partições de malha diferentes e que produz resultados idênticos em todas as partições, indepentemente da ordem na qual as entidades são tratadas ou da partição em que a operação ocorre.

Com base na representação de malha distribuída de ParTopS, operações topológicas simétricas são realizadas sobre entidades locais e *proxies*, de forma replicada em todas as partições que possuem uma cópia (local ou *proxy*) de uma determinada entidade. Considerando-se que os mesmos resultados são obtidos para a entidade em qualquer partição, não é necessária comunicação entre partições para garantir a consistência das operações. Assim, a comunicação para a atualização de entidades *proxies* é substituída pela replicação de operações topológicas nas partições. Apenas entidades *ghosts* são atualizadas, quando necessário, em relação às entidades locais ou *proxies* remotas correspondentes, através da comunicação entre partições vizinhas.

Com essa abordagem, operações topológicas que modificam uma entidade e, para isso, requerem a completude da vizinhança direta da entidade são executadas concorrentemente em partições diferentes, de forma consistente e sem requerer o acesso exclusivo à entidade por uma única partição. Da mesma forma, evita-se a migração temporária de entidades entre partições para se garantir a completude da vizinhança direta da entidade.

Uma vantagem de operações simétricas é o tratamento *"lock-free"* fornecido para a atualização das entidades topológicas. Como o acesso exclusivo a uma entidade não é necessário, elimina-se o uso de *travas de acesso exclusivo* (ou *locks*), comumente empregadas para proteger o acesso às entidades compartilhadas. Como consequência, em geral reduz-se o tempo de espera em cada partição para acessar a entidade e limita-se a quantidade de mensagens trocadas entre partições. Embora o emprego de travas de acesso possa se mostrar conveniente na implementação de outros operadores paralelos para malhas adaptativas, nesta pesquisa elas não se mostraram necessárias ao tratamento da inserção de elementos coesivos.

3.2.2. Descrição do algoritmo paralelo

A aplicação cliente é responsável pela identificação do conjunto de facetas fraturadas onde elementos coesivos serão inseridos. Esse conjunto é então passado como entrada do algoritmo de inserção paralela de elementos coesivos. Devido às características simétricas da abordagem proposta, o conjunto de facetas fraturadas em cada partição inclui tanto facetas locais como *proxies*, de maneira consistente entre todas as partições que compartilham uma mesma faceta. Isso significa que, se uma faceta é fraturada em uma partição, então ela

é fraturada em todas as partições que possuem uma cópia local ou *proxy* dela; caso contrário, ela não é fraturada em nenhuma partição. As facetas fraturadas utilizadas para ilustrar o algoritmo proposto são mostradas na Figura 22.

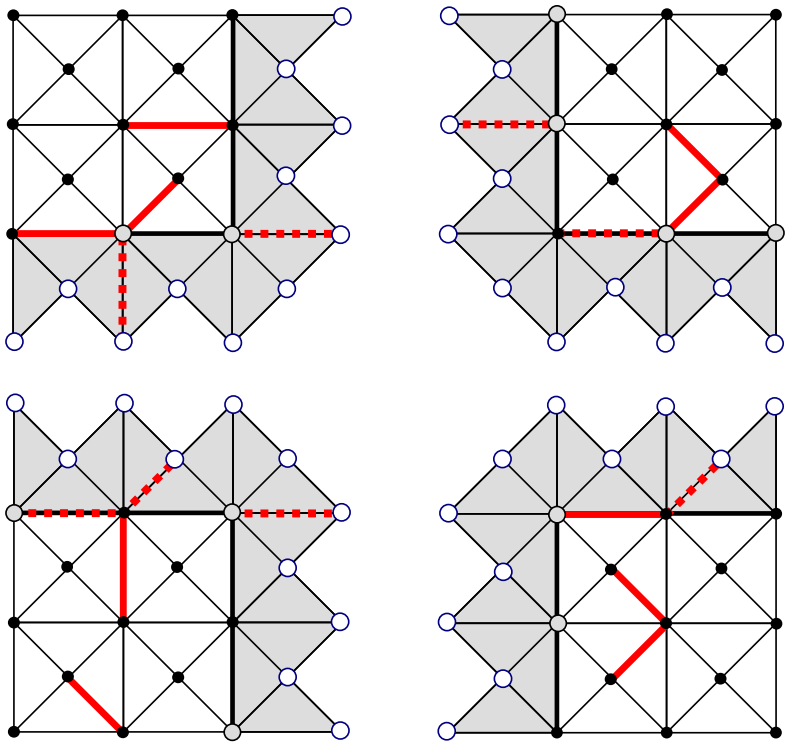

Figura 22 - Exemplo de malha distribuída, utilizada para demonstrar o algoritmo para a inserção adaptativa de elementos coesivos em paralelo. As facetas fraturadas (determinadas pela aplicação numérica) são destacadas.

Uma vez que as facetas fraturadas são identificadas, a inserção adaptativa de elementos coesivos em paralelo é executada. O algoritmo proposto consiste de três fases:

Fase 1. *Elementos coesivos são inseridos em facetas locais e proxies de cada partição, utilizando-se um algoritmo sequencial que produz resultados topológicos simétricos para cada faceta, em qualquer partição.*

Fase 2. *As referências de entidades proxies criadas, elementos e nós, são atualizadas com relação às entidades reais correspondentes.*

Fase 3. *As entidades ghosts afetadas pela inserção dos novos elementos coesivos são atualizadas em relação às entidades reais correspondentes.*

As três fases do algoritmo são detalhadas nas seguintes subseções.

3.2.2.1. Inserção sequencial de elementos coesivos (Fase 1)

Na primeira fase do algoritmo (Fase 1), elementos coesivos são inseridos nas facetas fraturadas locais e *proxies* de cada partição, com base na classificação topológica de facetas de Paulino et al. (2008), discutida na Seção 2.6.3. Segundo essa classificação, quando um novo elemento coesivo é adicionado à malha, pode ser necessário duplicar nós existentes. Neste ponto, faz-se a seguinte observação: suponhamos que os mesmos resultados topológicos possam ser obtidos para um conjunto de entidades em todas as partições de malha, independentemente da ordem em que elementos coesivos são inseridos localmente em cada partição, conforme a definição de operação simétrica (Seção 3.2.1). Então, a topologia de entidades locais e *proxies* será consistente para toda a malha de elementos finitos após a execução da Fase 1, sem a necessidade de comunicação entre partições para sincronizá-la. Com base na definição de malha distribuída de ParTopS (Seção 3.1), entretanto, nós do tipo *ghost* não são duplicados neste momento, devido à incompletude da vizinhança direta das entidades. Eles serão atualizados separadamente durante a Fase 3 do algoritmo.

Para garantir um comportamento topológico simétrico entre as partições da malha distribuída, adicionam-se duas restrições à definição original da classificação de facetas de Paulino et al. (2008). A primeira requer que, em qualquer partição onde um nó duplicado esteja presente, as cópias nodais correspondentes possuam referência a um mesmo elemento adjacente (em TopS, e assim ParTopS, todo nó possui uma referência a um dos elementos adjacentes a ele - relação *nó-a-elemento* (Seção 2.6)). Isso pode ser obtido com o emprego de um critério uniforme para a seleção do elemento referenciado. Por simplicidade, pode-se escolher, dentre os elementos adjacentes ao nó no momento da inserção de um elemento coesivo, o que possui o menor identificador global de elemento: (*owner_partition, owner_handle*) – Seção 3.1. Identificadores globais são comparados em ordem lexicográfica, primeiramente pela partição proprietária (*owner_partition*) da entidade, e em seguida pelo identificador local da entidade naquela partição (*owner_handle*).

A segunda restrição requer que todas as cópias de um novo nó ou elemento coesivo sejam atribuídas, consistentemente, à mesma partição

proprietária. No caso de um nó, essa restrição é obtida atribuindo-se o nó à partição proprietária do elemento ao qual ele faz referência. Uma vez que todas as cópias do nó possuem referência para o mesmo elemento real (conforme a primeira restrição), então a partição proprietária do nó é naturalmente consistente para todas as suas cópias. Elementos coesivos são atribuídos à mesma partição de um dos dois elementos volumétricos adjacentes a eles. Neste caso, o elemento volumétrico representativo é escolhido com base no mesmo critério uniforme utilizado na primeira restrição (menor valor do par (*owner_partition, owner_handle*)). As âncoras que definem entidades implícitas (Seção 2.6.1) também são atualizadas com base no mesmo procedimento.

A Fase 1 do algoritmo é ilustrada na Figura 23, para duas partições vizinhas. Três elementos coesivos são inseridos ao redor de um nó presente em ambas as partições (Figura 23a). Embora a ordem de inserção dos elementos seja diferente em cada partição, os mesmos resultados topológicos são obtidos para todas as entidades locais e *proxies* envolvidas na operação (Figura 23b). Note que os mesmos elementos são referenciados pelos novos nós criados, independentemente da partição de malha (indicados pela marca "x" na Figura 23), como resultado das restrições adicionais à classificação de facetas.

Considerando-se a definição de vizinhança de partições de ParTopS (Seção 3.1.3), as operações topológicas realizadas não alteram o conjunto de vizinhança de cada partição, pois as entidades criadas são atribuídas a partições já pertencentes ao conjunto. Com isso, não é necessário que uma partição se comunique com outras a fim de atualizá-lo.

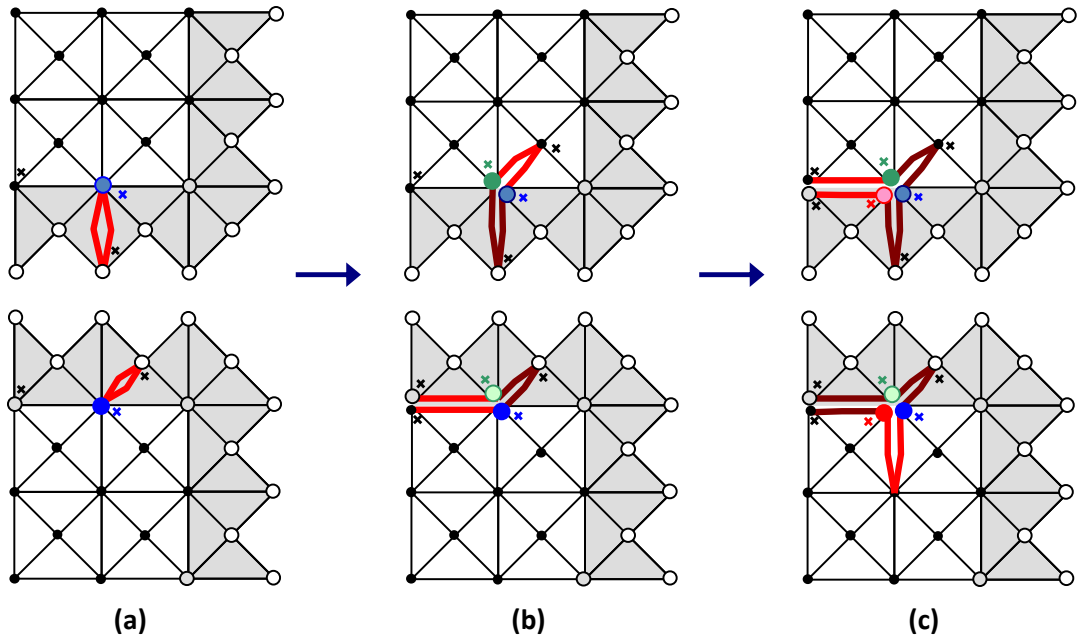

Figura 23 - Fase 1 da inserção de elementos coesivos. Três elementos são inseridos (a - c) em facetas de duas partições vizinhas. Embora a ordem de inserção seja diferente, os mesmos resultados topológicos são obtidos para as entidades locais e *proxies* de ambas as partições (c). As marcas 'x' identificam o elemento referenciado por cada nó modificado. Nós *ghosts* não são duplicados; eles serão atualizados na Fase 3.

3.2.2.2. Atualização das entidades *proxies* criadas (Fase 2)

Ao final da Fase 1, a topologia de entidades locais e *proxies* encontra-se consistente entre todas as partições de malha. Entretanto, as referências das novas entidades *proxies* criadas (elementos coesivos e nós duplicados) para as entidades reais correspondentes ainda precisam ser determinadas (Figura 24). Referências para entidades reais fazem parte da definição de malha distribuída de ParTopS (conforme Seção 3.1.1), e devem estar consistentes para que futuras operações topológicas e a sincronização de atributos de simulação possam ser realizadas. A partição proprietária de uma nova entidade *proxy* foi determinada durante a Fase 1, uma vez que ela corresponde à partição de um dos elementos adjacentes à entidade. Porém, o identificador local da entidade com respeito à partição proprietária não é conhecido pela partição corrente. Essa informação deve ser requisitada à partição proprietária.

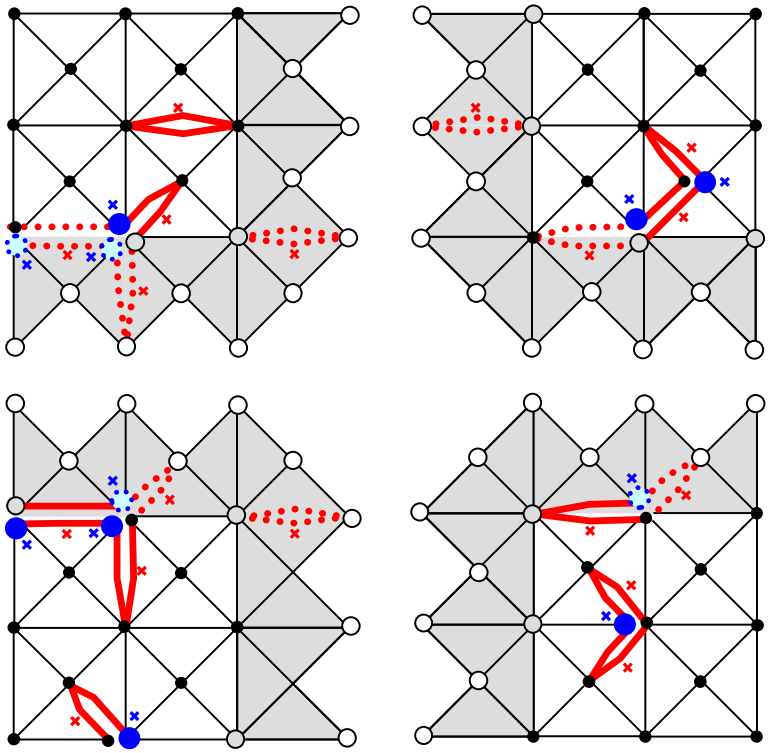

Figura 24 - Malha completa após a Fase 1. A partição escolhida como proprietária de um novo elemento coesivo ou nó duplicado é a mesma de um dos elementos volumétricos adjacentes ao coesivo (indicado pelas marcas 'x'). Os elementos coesivos estão destacados na figura. Os *proxies* correspondentes são representados por linhas tracejadas. Embora a topologia de entidades locais e *proxies* esteja consistente neste ponto, as referências para as entidades reais ainda devem ser computadas.

Dessa forma, a Fase 2 do algoritmo paralelo para a inserção de elementos coesivos é responsável pela atualização das referências às entidades reais correspondentes dos novos elementos coesivos e nós duplicados do tipo *proxy*. Isso pode ser feito da seguinte forma: uma lista de requisições de referências é enviada pela partição corrente a cada partição vizinha a ela. As partições vizinhas então acessam as entidades reais correspondentes e retornam os respectivos identificadores locais. Finalmente, a partição corrente atualiza as referências das novas entidades *proxies* com os valores recebidos.

É importante notar que as próprias entidades *proxies* não podem ser usadas na requisição das referências que lhes faltam. Entretanto, observa-se também que qualquer elemento (coesivo ou volumétrico) pode ser univocamente identificado por um dos elementos adjacentes a ele. Isso é expresso pela tripla ($owner\_partition_{adj}$, $owner\_element\_handle_{adj}$, *local_id*), onde $owner\_partition_{adj}$ é

o identificador da partição proprietária do elemento adjacente, $owner\_element\_handle_{adj}$ é o identificador do elemento adjacente naquela partição, e *local_id* é o índice da faceta-uso incidente ao novo elemento coesivo *proxy,* na ordem de incidência fixa do elemento adjacente (Seção 3.1.1). De maneira equivalente, um nó *proxy* também pode ser identificado por um dos elementos adjacentes. Com isso, elementos adjacentes são utilizados para obter as referências faltantes das entidades *proxies* criadas.

Como consequência da Fase 1 do algoritmo, a partição proprietária de um elemento coesivo corresponde à partição proprietária de um dos elementos volumétricos adjacentes a ele. O elemento adjacente representativo é, então, usado para a requisição da referência do elemento coesivo *proxy* à entidade real correspondente. Da mesma forma, utiliza-se o elemento adjacente referenciado por um novo nó *proxy* na requisição da referência do nó à entidade real correspondente, considerando-se que, durante a Fase 1, o nó real foi atribuído à mesma partição do elemento.

As mensagens trocadas entre partições vizinhas para atualizar as referências das entidades *proxies* criadas são ilustradas na Figura 25. A comunicação é iniciada pela partição corrente (l*ocal*). A partição envia uma mensagem consistindo de pares ($owner\_element\_handle_{adj}$, *local_id*) a cada partição vizinha (*remota*) que é proprietária ($owner\_partition_{adj}$) dos elementos adjacentes representativos das entidades *proxies* a serem atualizadas. A mensagem de retorno contém os identificadores locais das entidades reais nas partições proprietárias correspondentes.

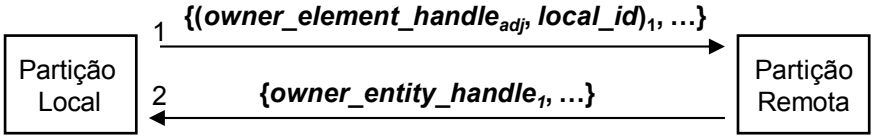

Figura 25 - Diagrama de mensagens trocadas entre duas partições para a atualização das referências de entidades *proxies*.

3.2.2.3. Atualização das entidades *ghosts* afetadas (Fase 3)

A última fase (Fase 3) do algoritmo paralelo é responsável pela atualização de nós do tipo *ghost* afetados por duplicações de nós ocorridas em outras partições, resultantes da inserção dos novos elementos coesivos (Figura 26).

Devido à incompletude da vizinhança direta de nós do tipo *ghost*, as duplicações nodais não podem ser determinadas localmente por uma partição. Por exemplo, um elemento coesivo pode ter sido inserido em uma faceta remota que não é conhecida pela partição corrente, mas que implica na duplicação do nó real que corresponde a uma entidade *ghost* da partição. Na Figura 26, os nós *ghosts* afetados pela duplicação de nós em outras partições são mostrados. Para manter a topologia da malha da partição corrente consistente com a malha global distribuída, pode ser necessário ao procedimento de atualização de *ghosts* substituir a referência do nó *ghost* para uma nova entidade real correspondente, ou mesmo duplicá-lo em novos nós *ghosts* distintos.

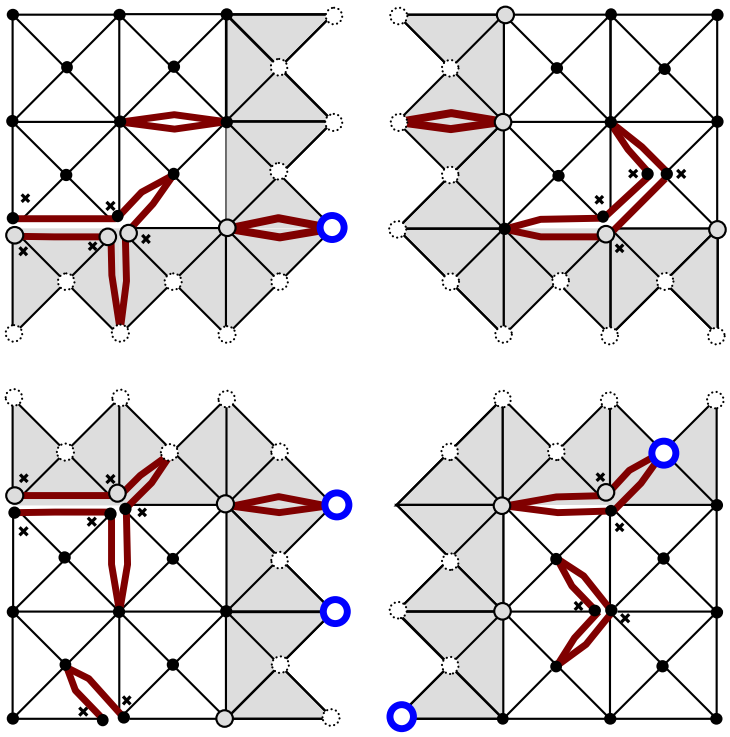

Figura 26 - Malha completa antes do início da Fase 3. Os nós *ghosts* de cada partição afetados pela duplicação de nós ocorridas em outras partições são destacados na figura. Esses nós precisam ser atualizados.

Como a partição corrente não pode determinar exatamente os nós *ghost* afetados, que devem ser atualizados, uma forma simples de se garantir a consistência topológica de nós *ghosts* consiste na atualização de todos em relação às entidades reais correspondentes. Porém, é comum que, a cada passo da simulação de fraturas, um pequeno número de elementos coesivos seja adicionado à malha distribuída, em relação ao número total de elementos de

cada partição. Assim, é desejável uma abordagem proporcional ao número de entidades afetadas.

Para isso, uma partição contendo nós locais ou *proxies,* cuja duplicação na Fase 1 tenha afetado nós *ghosts* remotos, deve notificar as outras sobre as alterações ocorridas. Contudo, a representação de malha distribuída de ParTopS não prevê que entidades locais mantenham referências para as suas cópias *proxies* e *ghosts*, apenas no sentido inverso. Isso evita o custo de armazenamento e manutenção de listas explícitas de referências dinâmicas. Por outro lado, requer que a comunicação relacionada a uma entidade real seja iniciada pelas partições que possuem *proxies* ou *ghosts* dela.

No contexto da inserção de elementos coesivos, e a fim de se evitar a necessidade de listas dinâmicas, optou-se por uma abordagem para a atualização de nós *ghosts* baseada em elementos. Nela, em vez de cada nó local de uma partição armazenar referências para entidades *ghosts* remotas, são os elementos locais que possuem referências às partições contendo elementos *proxies* correspondentes. Com isso, os elementos locais são responsáveis por informar a outras partições sobre alterações em nós adjacentes que potencialmente sejam representados por elas como *ghosts*. A vantagem dessa abordagem é que referências existentes para elementos *proxies* não precisam ser atualizadas quando novos elementos coesivos são inseridos na malha distribuída, pois nenhum elemento *proxy* é substituído ou removido da malha durante esse processo. Assim, não é necessário manter listas de referências dinâmicas. Também, a vizinhança de cada partição não precisa ser recomputada, considerando-se a definição da Seção 3.1.3, pois a atualização de atributos de entidades *ghosts* pode ser feita a partir dos elementos *proxies* adjacentes (Seção 3.1.3).

Para mostrar que a atualização baseada em elementos pode ser empregada de forma a garantir a consistência de nós *ghosts*, observa-se o seguinte. Um nó *ghost* faz parte da camada de comunicação de uma partição. Logo, ele deve ser adjacente a pelo menos um elemento do tipo *proxy*, e todo elemento *proxy* corresponde a um elemento local na sua partição proprietária. O elemento local é adjacente a nós locais ou *proxies*, sendo que um deles corresponde ao nó *ghost* considerado. Pela simetria imposta na Fase 1 do algorimo, qualquer duplicação de um nó local ou *proxy* é consistente entre todas as partições da malha. Então, se o nó real correspondente ao *ghost* tiver sido duplicado em alguma partição, ele também será na partição proprietária do elemento *proxy* adjacente ao nó *ghost*. Com isso, o nó *ghost* será afetado pela

duplicação nodal se e somente se o elemento local na partição proprietária do elemento *proxy* adjacente ao nó *ghost* tiver o nó local ou *proxy* correspondente afetado.

O procedimento para a atualização da topologia de entidades *ghosts* é descrito a seguir. Inicialmente, para cada nó duplicado na partição corrente, determinam-se os elementos locais que são adjacentes a ele e que possuem elementos *proxies* em outras partições (Figura 27). Entre os elementos locais adjacentes ao nó, vários deles podem possuir *proxies* em uma mesma partição remota. Porém, apenas um por partição é selecionado. Em seguida, uma mensagem é enviada às partições dos elementos *proxies* correspondentes, a fim de notificá-las sobre as duplicações nodais. A mensagem consiste de tuplas (*owner_element_handle*, *local_id*), onde *owner_element_handle* é o identificador de um elemento local, e *local_id* é o índice de um nó na ordem de incidência fixa do elemento (Figura 28). Quando a notificação é recebida por uma partição, cada identificador de elemento é mapeado para o elemento *proxy* correspondente, e o nó relativo ao índice *local_id* é acessado. Se o nó for do tipo *ghost*, então ele é marcado como *desatualizado* (*outdated*) (Figura 27).

Para cada elemento *proxy* adjacente a um nó *ghost* desatualizado (Figura 29), uma requisição de dados nodais é criada. A requisição consiste da tupla (*owner_element_handle, local_id*), onde *owner_element_handle* é o identificador local do elemento em sua partição proprietária, e *local_id* é o índice do nó na ordem de incidência fixa do elemento (Seção 3.1.1). Então, o conjunto de requisições é enviado às partições proprietárias dos elementos *proxies* (Figura 28). Note que mais de uma requisição para um mesmo nó *ghost* pode ser emitida, quando há vários elementos *proxies* adjacentes a ele. Entretanto, isso garante que a duplicação de nós seja resolvida de maneira consistente para cada elemento.

Finalmente, a partição proprietária de cada elemento *proxy* retorna os dados nodais requisitados (Figura 28), e os nós *ghosts* são atualizados. Os dados de um nó consistem de valores (*owner_node_handle, (x, y, z)*), onde *owner_node_handle* é o identificador do nó na sua partição proprietária, e (*x, y, z*) é a posição do nó. Embora um nó *ghost* possa corresponder a um nó do tipo *proxy* na partição proprietária do elemento *proxy* adjacente a ele, o nó *proxy* já foi atualizado durante a Fase 2, e, assim, os seus dados encontram-se consistentes. Após a atualização de nós *ghosts*, as âncoras (Seção 2.6.1) que definem entidades implícitas do tipo *ghost* são ajustadas de forma a se manter a

consistência topológica local da partição corrente. A malha final é mostrada na Figura 30.

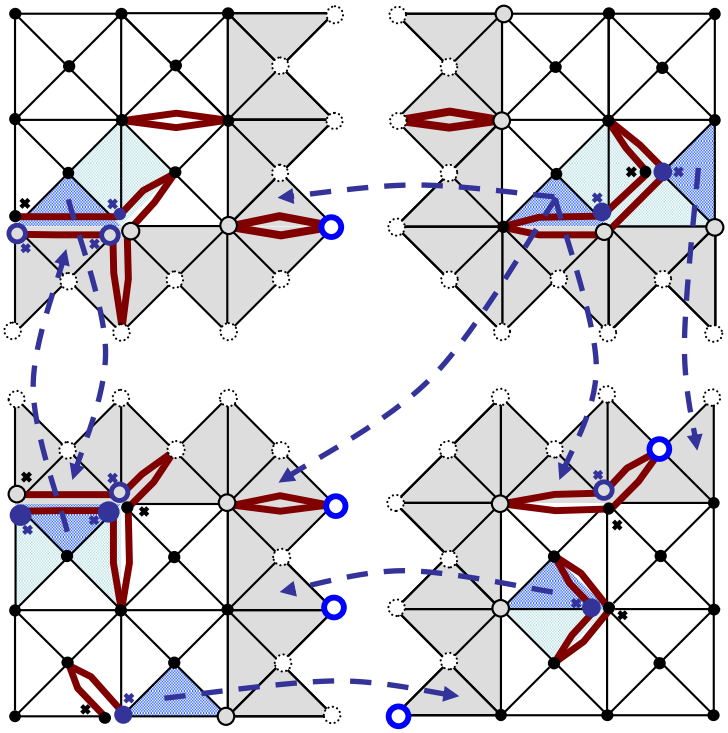

Figura 27 - Elementos locais que são incidentes a nós duplicados e têm elemento *proxy* correspondente em outra partição são destacados. Esses elementos notificam partições vizinhas sobre duplicações nodais que possam afetar nós *ghosts*. Porém, apenas um elemento por nó duplicado e partição vizinha é necessário (os de origem das setas tracejadas). Nós *ghosts* de elementos *proxies* notificados são marcados como *desatualizados*. Os nós *ghosts* destacados com bordas contínuas devem ser atualizados.

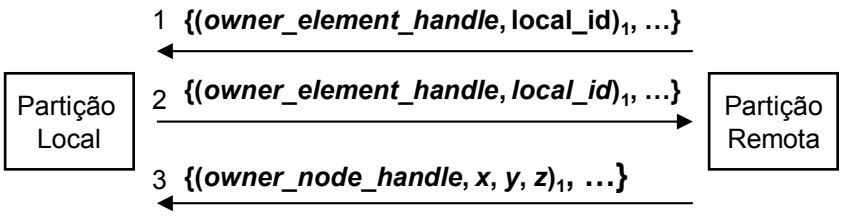

Figura 28 - Diagrama de mensagens trocadas entre a partição corrente (local) e uma vizinha a ela (remota), para atualizar entidades *ghosts* afetadas pela duplicação de nós ocorrida na partição vizinha.

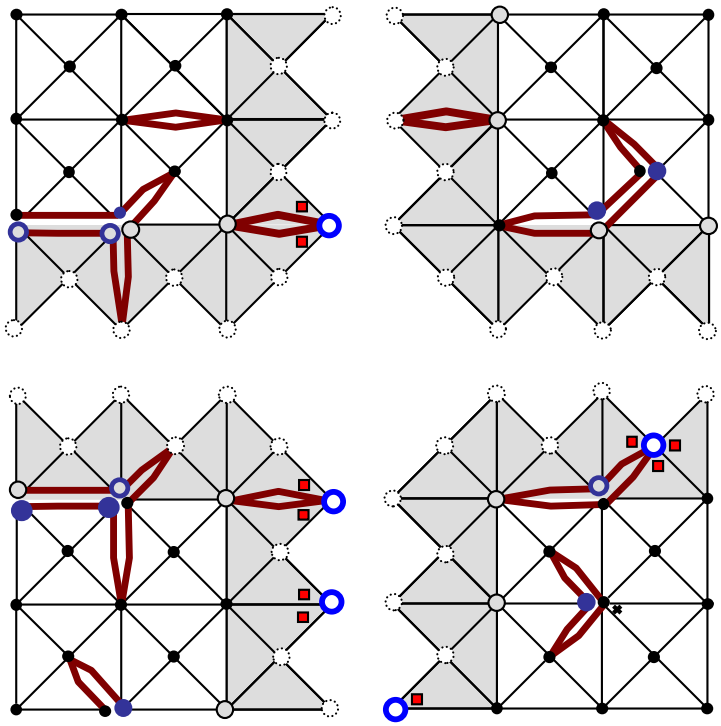

Figura 29 - Para cada elemento *proxy* incidente a nó *ghost desatualizado*, uma requisição do nó real correspondente é enviada à partição proprietária do elemento. Requisições nodais são indicadas por marcas quadradas ao lado de nós *ghosts*.

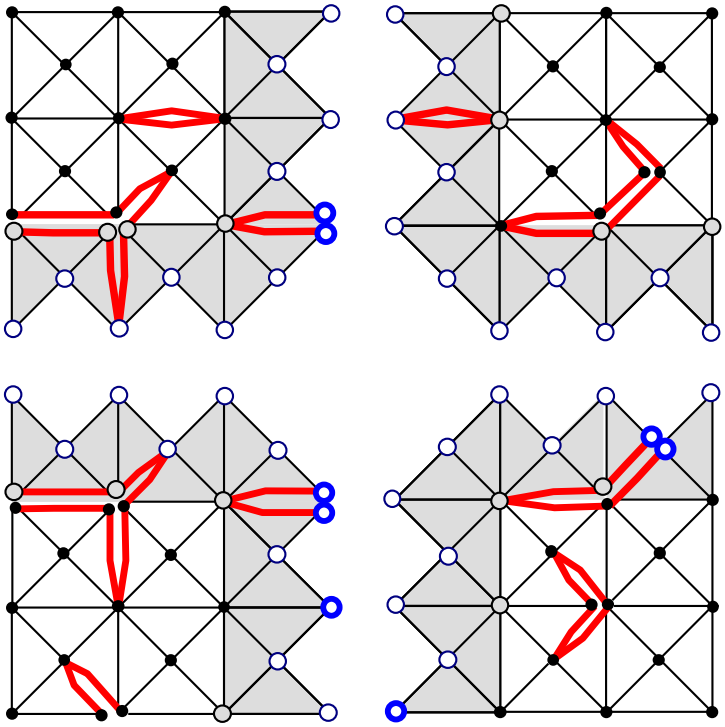

Figura 30 - Malha final após a inserção paralela de elementos coesivos. A topologia da malha distribuída encontra-se consistente.

A implementação da abordagem de atualização de nós *ghosts* baseada em elementos adjacentes segue a filosofia de representação compacta de ParTopS. Com isso, em vez de armazenar o conjunto de todas as partições que possuem *proxies* para um determinado elemento local, um valor inteiro é empregado para indicar a existência de *proxies* do elemento. Assim, quando há uma única partição que contém um *proxy* para o elemento, o que corresponde ao caso mais comum, o valor inteiro é igual ao identificador da partição. Se houver duas ou mais partições, um valor negativo é usado como contador para indicar o número de partições. Nesse caso, notificações de duplicações nodais relacionadas ao elemento são enviadas a todas as partições vizinhas. Finalmente, se nenhuma partição possui um *proxy* para o elemento, o valor inteiro é igual a zero.

3.2.3. Interface para a inserção de elementos coesivos

O suporte à inserção paralela de elementos coesivos é fornecido à aplicação cliente através de uma função paralela coletiva, executada concorrentemente por cada partição. A função recebe como parâmetros: o tipo dos elementos coesivos a serem inseridos e o conjunto de facetas fraturadas na partição corrente, incluindo tanto as locais quanto *proxies*. Do ponto de vista da aplicação cliente, a função sequencial original, de assinatura similar, é substituída pela versão paralela.

A função paralela não retorna os elementos coesivos que foram inseridos; em vez disso, a aplicação é notificada sobre cada elemento coesivo criado ou nó duplicado, por *funções de chamada de retorno* (*callback functions*) - ou *funções de chamada de volta* - registradas na representação de malha distribuída. A interface de chamadas de retorno de duplicações nodais é a mesma de TopS, mas as funções também são chamadas para entidades *proxies* e *ghosts*. Quando notificada, a aplicação deve criar os atributos relativos aos novos nós e elementos. A notificação de duplicações de nós locais e *proxies* ocorre durante a Fase 1 do algoritmo paralelo, enquanto que atributos de nós *ghosts* afetados são atualizados na Fase 3. Ao final do algoritmo, a aplicação é notificada sobre os elementos coesivos criados.

3.2.4. Análise de escalabilidade

A *escalabilidade* de um sistema paralelo (Foster, 1995; Quinn, 2004) está relacionada à capacidade de se utilizar de maneira eficiente uma quantidade maior de recursos para resolver problemas maiores ou mais complexos. Diversas abordagens existem para a medição de escalabilidade em diferentes circunstâncias (Foster, 1995; Quinn, 2004; Kumar & Gupta, 1994). Nesta seção, a métrica de *isoeficiência* (Grama et al., 1993) é empregada para avaliar a escalabilidade do algoritmo proposto. Essa métrica relaciona o tamanho do problema paralelo com o número de processadores necessários para se manter a eficiência de um sistema. Sem perda de generalidade, um modelo bidimensional é usado como exemplo. Resultados similares podem ser obtidos para modelos tridimensionais.

Considere o tempo de execução $T(n, p)$ de um sistema paralelo com tamanho de problema n para p processadores. O tempo sequencial é $T_1 = T(n, 1)$, e $T_0 = T_0(n, p)$ é o tempo adicional introduzido devido à implementação paralela. Então, a *eficiência parelela* do sistema (Grama et al., 1993) é definida como $E = 1 / (1 + T_0/T_1)$; ou de forma equivalente: $T_1 = (E / (1 - E)).T_0 = K.T_0$.

Para se manter a eficiência, o tempo de computação sequencial T_1 deve aumentar a uma taxa maior ou igual ao tempo adicional paralelo T_0, ou seja, $T_1 \geq K.T_0$. Paulino et al. (2008) demonstram a escalabilidade linear da inserção sequencial de elementos coesivos, em função do número de elementos inseridos. Dessa forma, o tempo da execução sequencial pode ser expresso por: $T_1 = n.t_c$, onde n é o número de elementos coesivos inseridos, e t_c é o custo médio da operação de inserção de um elemento coesivo. Assim, a eficiência paralela é mantida quando $n \geq C.T_0$, onde $C = K / t_c$.

No algoritmo paralelo de inserção de elementos coesivos, o custo adicional paralelo (T_0) se deve à replicação de operações topológicas na camada de comunicação e à comunicação entre partições vizinhas, como ilustrado na Figura 31. Observa-se que a replicação de operações topológicas é proporcional ao tamanho da camada de comunicação, em média equivalente a $\sqrt{n}/\sqrt{p}$. Por sua vez, o número de mensagens entre partições vizinhas durante a execução do algoritmo é constante, e o tamanho de cada mensagem proporcional ao tamanho da camada de comunicação. Com isso, o custo paralelo de um processador pode ser expresso por: $T_p \approx c_1(\sqrt{n}/\sqrt{p})t_c + c_2 t_s + c_3(\sqrt{n}/\sqrt{p})t_w$, onde c_1, c_2, e c_3 são multiplicadores constantes, t_s é o tempo médio de inicialização da

comunicação para o envio de uma mensagem, e t_w é o tempo médio para se enviar uma unidade de dados. O custo paralelo total (para todos os procesadores) é então: $T_0 = pT_p = c_1 t_c \sqrt{np} + c_2 t_s + c_3 t_w \sqrt{np}$. Nesse modelo, assume-se que a carga dos processadores é balanceada entre eles, e, assim, os tempos ociosos (*idle times*) de cada processador não são significativos.

Para um grande número de processadores, a eficiência paralela é mantida quando o crescimento assintótico do tamanho do problema *n* é maior ou igual ao de T_0. Considerando-se que o crescimento de T_0 é proporcional a $\sqrt{np}$, obtém-se a seguinte relação: $n \geq C\sqrt{np}$, que leva a: $n \geq C^2 p$. Logo, quando *n* cresce proporcionalmente a *p*, mantêm-se a eficiência do algoritmo paralelo, e, assim, a escalabilidade linear em relação ao número de processadores é esperada.

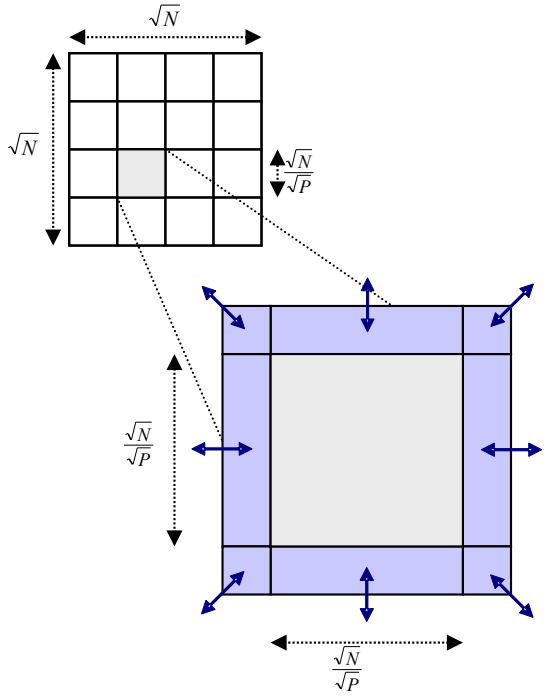

Figura 31 - Representação de malha bidimensional de dimensões *N x N*, decomposta em *P* partições. O padrão de comunicação correspondente a uma partição é destacado.