4
Simulações de fraturas extrínsecas em ambientes paralelos

Este capítulo descreve a utilização do suporte topológico ParTopS no desenvolvimento análises numéricas em paralelo. Como exemplo, discute-se a paralelização de uma aplicação sequencial existente para simulações de fraturas dinâmicas. A aplicação, originalmente implementada com base na representação de malha sequencial de TopS (Celes, Paulino & Espinha, 2005a, b; Paulino et al., 2008), é paralelizada pela introdução do conjunto de extensões paralelas fornecidas por ParTopS (Espinha et al., 2009). Um número pequeno de modificações é necessário ao código original para que a análise numérica seja executada em um ambiente paralelo de memória distribuída.

4.1.
Requisitos de sincronização de entidades

Uma característica de análises dinâmicas de elementos finitos é a interdependência de dados. A computação de um passo de tempo da simulação numérica depende de resultados do passo anterior. Da mesma forma, computações intermediárias de um passo também dependem de resultados anteriores. Isso restringe a capacidade de paralelização de uma simulação, uma vez que a computação de dados dependentes não pode ser realizada concorrentemente. Pontos de sincronização ocorrem quando os dados requeridos a uma computação ainda não se encontram disponíveis à partição corrente, que deve aguardar até que eles sejam recebidos. Com isso, as oportunidades de paralelização da simulação residem na decomposição dos dados do domínio geométrico, que podem ser processados de forma concorrente entre os pontos de sincronização. Como exemplo, a computação de tensões nodais, utilizada na determinação de facetas fraturadas, requer que os deslocamentos nodais tenham sido previamente computados. Considerando que os deslocamentos encontram-se consistentes em cada partição, as tensões nodais podem então ser calculadas concorrentemente por cada partição.

Uma partição de malha corresponde a uma unidade sequencial de computação. No interior dela, a computação numérica é realizada de maneira independente a partir dos dados locais à partição. Porém, atributos de simulação associados a entidades nas fronteiras compartilhadas com outras partições devem se manter consistentes entre as partições envolvidas. Esses atributos são utilizados na computação dos resultados das próprias entidades e de outras adjacentes a elas. Com isso, a comunicação entre partições vizinhas pode se tornar necessária para sincronizar os dados correspondentes.

A camada de comunicação adicionada por ParTopS provê as entidades adjacentes necessárias a computações locais em entidades das fronteiras originais de cada partição de malha. Para se manter a consistência dos dados das entidades da camada de comunicação com as partições vizinhas, duas classes de abordagens são avaliadas. A primeira corresponde à estratégia convencional, na qual entidades da camada de comunicação são utilizadas apenas para a leitura de dados, e os resultados de uma computação são armazenados somente nas entidades locais de cada partição (note que a computação pode usar dados fornecidos por entidades *proxies* adjacentes). Com isso, os resultados de uma entidade são determinados exclusivamente por uma única partição (a proprietária da entidade), correspondendo a um mecanismo implícito de *travas de acesso exclusivo* (*locks*) (Andrews, 2000) à entidade. Considerando-se que os dados das entidades adjacentes às locais encontram-se disponíveis na partição corrente, por meio da camada de comunicação, a computação pode proceder de forma sequencial, sem problemas de concorrência para a atualização de entidades. Por outro lado, as entidades da camada de comunicação devem ser sincronizadas com as entidades reais correspondentes sempre que dados modificados precisarem ser utilizados.

A segunda classe de abordagens explora as propriedades da camada de comunicação definida por ParTopS. Dessa forma, as entidades da camada de comunicação são consideradas editáveis, e resultados de computações são calculados e armazenados nelas, de forma replicada com as entidades reais correspondentes, sempre que conveniente à aplicação. Uma vez que dados associados a uma mesma entidade podem ser computados simultaneamente por partições de malha diferentes, é preciso garantir que eles se encontrem consistentes entre elas. Para isso, propõe-se o uso de computações *simétricas* entre as partições, de maneira similar às operações topológicas empregadas pelo algoritmo paralelo de inserção de elementos coesivos do Capítulo 3. Com isso, o número de pontos de sincronização de dados entre partições é reduzido e

a estrutura da simulação paralela simplificada. A simetria de computações é discutida na Seção 4.4.

Note que não é preciso atualizar todos os atributos de uma entidade topológica em cada ponto de sincronização, mas somente aqueles que devem se encontrar consistentes para a computação seguinte. A sincronização pode ser postergada até que os dados sejam realmente necessários à simulação, ou realizada de maneira assíncrona, enquanto computações independentes são executadas. No entanto, a aplicação deve bloquear a execução caso os dados requeridos não estejam disponíveis no momento esperado.

4.2. Estrutura da simulação sequencial

A estrutura do fluxo de controle básico de uma simulação numérica de fraturas dinâmicas, para pequenas deformações, é resumida na Figura 32; a aplicação numérica sequencial correspondente foi desenvolvida por Park (2009). A integração temporal é baseada no método de diferenças centrais (método explícito) (Belytschko et al., 2000), e o modelo constitutivo *Park-Paulino-Roesler (PPR)* (Park et al., 2009; Park, 2009) é utilizado com um modelo de zona coesiva extrínseco (Camacho & Ortiz, 1996; Ortiz & Pandolfi, 1999; Park et al., 2009). A estrutura da simulação sequencial apresentada na Figura 32 é utilizada como base da aplicação paralela discutida nas próximas seções.

```
Inicialize o modelo de elementos finitos

Para cada passo de tempo
  1 - Compute o valor das condições de contorno
  2 - Verifique a inserção de novos elementos coesivos
    2.1 - Compute as tensões nodais
    2.2 - Determine as facetas fraturadas
    2.3 - Insira elementos coesivos
    2.4 - Compute atributos de novos elementos coesivos
    2.5 - Atualize os valores das massas nodais
  3 - Compute o vetor de forças nodais internas
  4 - Compute o vetor de forças coesivas
  5 - Atualize as acelerações e velocidades nodais
  6 - Atualize as forças externas e energias nos nós
  7 - Aplique as condições de contorno aos nós
  8 - Atualize os deslocamentos nodais
```

Figura 32 - Estrutura do fluxo de controle de uma simulação sequencial de propagação dinâmica de fraturas.

A simulação da propagação de fraturas ocorre ao longo de um número de passos de tempo discretos (Figura 32). Em cada passo, a aplicação numérica deve determinar as facetas que serão fraturadas e inserir os elementos coesivos correspondentes (linha 2). A determinação de facetas fraturadas é feita com base nos valores de tensão (*stress*) dos nós adjacentes; assim, as tensões nodais devem ser primeiramente calculadas (linha 2.1). Os valores de tensão (*stress*) são computados nos pontos de Gauss de cada elemento volumétrico, a partir de atributos dos nós adjacentes, e então extrapolados novamente para os nós do elemento. As tensões nodais finais são obtidas pela média das contribuições dos elementos adjacentes a cada nó. Com as tensões dos nós adjacentes calculadas, as facetas internas da malha (interfaces entre elementos volumétricos) são avaliadas em relação a um critério de fratura (linha 2.2), e toda faceta para a qual o critério de fratura é alcançado é marcada como *fraturada* no passo corrente. Em seguida, elementos coesivos são criados *sob demanda* e inseridos nas facetas fraturadas (linha 2.3). Os atributos dos nós duplicados são copiados do nó anterior correspondente, através de *funções de chamadas de retorno* (*callbacks*), registradas pela aplicação e chamadas por TopS, para cada duplicação de nó. Atributos de novos elementos coesivos (linha 2.4) são calculados a partir das tensões associadas aos nós adjacentes. As massas de nós duplicados também são atualizadas; a massa de um nó corresponde ao somatório das contribuições de massa dos elementos volumétricos adjacentes.

Após as operações relativas à verificação da inserção de elementos coesivos (linha 2), computações adicionais são realizadas a fim de se atualizar o estado da simulação necessário ao próximo passo de tempo ou exportado como saída da simulação. O vetor de forças nodais internas (linha 3) é calculado a partir do somatório das contribuições de forças dos elementos volumétricos adjacentes a cada nó da malha de elementos finitos. A contribuição de um elemento é baseada na rigidez (*stiffness*) do elemento e deslocamentos dos nós adjacentes. Forças coesivas (linha 4), por outro lado, são calculadas apenas para nós adjacentes a elementos coesivos. Primeiramente, separações coesivas são computadas nos pontos de Gauss de cada elemento coesivo, a partir dos deslocamentos dos nós adjacentes. Então, as separações e outros atributos do elemento (ex. material e história de carregamento) são usados na computação de trações e forças correspondentes, e a contribuição do elemento é somada a cada nó adjacente. Finalmente (linhas 6 a 8), os valores restantes (acelerações, velocidades, energia, condições de contorno e deslocamentos nodais) são calculados em cada nó, com base nos atributos relativos aos próprios nós.

4.3. Padrões de computação

Quatro padrões de computação básicos, relacionados à origem e o destino de dados associados a entidades topológicas, podem ser identificados em uma análise de elementos finitos regular. São eles: *nó-a-nó*, *elemento-a-elemento*, *nós-a-elemento* e *elementos-a-nó*. Computações dos tipos *nó-a-nó* (Figura 33a) e *elemento-a-elemento* (Figura 33b) ocorrem de maneira local aos nós ou elementos correspondentes. Assim, dependem apenas dos dados das próprias entidades que modificam. No padrão *nós-a-elemento*, os resultados de um elemento são calculados a partir das contribuições de dados associados aos nós adjacentes a ele (Figura 33c). Uma vez que os dados nodais estejam consistentes, a computação ocorre de maneira local ao elemento destino. Por outro lado, no padrão *elementos-a-nó*, os dados correspondentes a um elemento contribuem para os resultados armazenados nos nós adjacentes a ele (Figura 33d). Quando uma estrutura topológica apropriada é empregada, os resultados nodais podem ser calculados percorrendo-se o conjunto de nós da malha de elementos finitos. Para cada nó, os elementos adjacentes são visitados e as contribuições dos elementos são calculadas e combinadas no nó. Com base na representação de malha convencional de elementos finitos (tabelas de nós e conectividades nodais de elementos), entretanto, esse padrão é tradicionalmente implementado percorrendo-se o conjunto de elementos da malha. Então, os dados correspondentes a cada elemento são calculados e adicionados aos nós adjacentes, com base na conectividade nodal do elemento.

Computações mais complexas podem depender de entradas mistas e gerar resultados segundo padrões diferentes. Porém, essas computações podem ser decompostas em outras computações menores baseadas nos padrões básicos, e cada parte analisada individualmente. A classificação em padrões de computação fornece uma maneira sistemática e abstrata para se identificarem os requisitos de sincronização de atributos para a manutenção da consistência de dados compartilhados entre partições de malha. Com isso, a necessidade de sincronização é determinada apenas a partir da estrutura da computação, de forma independente do entendimento da simulação numérica.

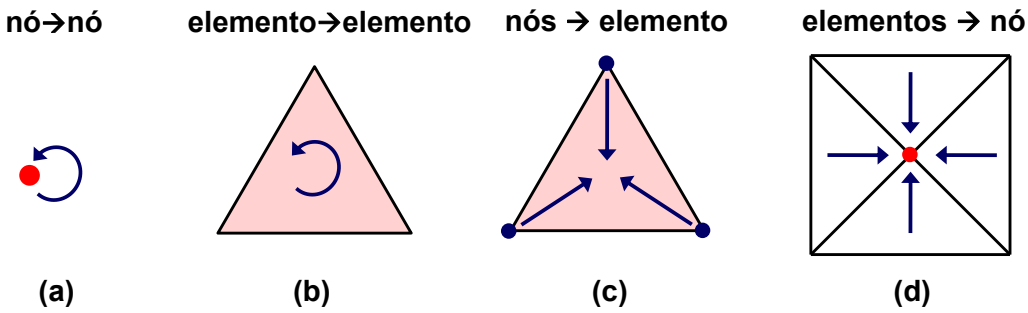

Figura 33 - Padrões de computação. (a) *nó-a-nó*: o resultado de um nó depende apenas dele próprio; (b) *elemento-a-elemento*: o resultado de um elemento depende apenas dele próprio; (c) *nós-a-elemento*: o resultado de um elemento depende dos nós adjacentes; (d) *elementos-a-nó*: o resultado de um nó depende das contribuições relativas aos elementos adjacentes.

4.4. Computações simétricas

Uma computação é definida como *simétrica* quando produz os mesmos resultados para uma entidade topológica compartilhada por partições diferentes, independentemente da partição onde é executada. Considerando-se que os resultados da computação são idênticos em qualquer partição, a comunicação entre partições para sincronizar os dados alterados da entidade topológica pode ser eliminada. Com isso, emprega-se conceito equivalente às operações topológicas discutidas na Seção 3.2.1, com a substituição da comunicação entre partições por computações replicadas, realizadas tanto para entidades locais quanto *proxies* e *ghosts* de cada partição.

Entretanto, operações binárias de ponto flutuante, como adição e multiplicação, não são associativas (Heath, 2002) e, assim, resultados diferentes podem ser obtidos em função da ordem em que as operações são realizadas. Dessa forma, computações executadas concorrentemente por partições diferentes para uma mesma entidade topológica podem produzir resultados divergentes, se a ordem das operações de ponto flutuante não for idêntica em todas as partições. As divergências numéricas, mesmo que pequenas, podem fazer com que decisões diferentes sejam tomadas por cada partição, por exemplo, se uma faceta deve ser fraturada. Essas divergências também podem se propagar ao longo dos passos de simulação, afetando a convergência da simulação.

Se pudermos garantir uma ordenação consistente das operações de ponto flutuante, então resultados idênticos (ou simétricos) serão obtidos para uma entidade em todas as partições, assumindo-se que a mesma representação de ponto flutuante é utilizada por elas. Para isso, primeiramente observam-se as características de simetria dos padrões de computação discutidos na seção anterior. Computações dos tipos *nó-a-nó* e *elemento-a-elemento* são naturalmente independentes de ordem e, assim, simétricas, uma vez que requerem apenas o acesso local à própria entidade. No padrão *nós-a-elemento*, os resultados de um elemento dependem da ordem em que as contribuições dos nós adjacentes são combinadas. Contudo, a ordenação nodal é imposta pela topologia local fixa do elemento, que é consistente entre todas as partições que possuem uma cópia da entidade. Dessa forma, computações do tipo *nós-a-elemento* também são simétricas. Por outro lado, em uma computação do tipo *elementos-a-nó,* os resultados de um nó dependem da ordem em que as contribuições dos elementos adjacentes são combinadas. As abordagens para computações *elementos-a-nó* simétricas são discutidas a seguir.

4.4.1. Computações elementos-a-nó simétricas com iteradores estáveis

Resultados simétricos de computações do tipo *elementos-a-nó* podem ser obtidos de duas formas. Na primeira, as contribuições fornecidas por elementos adjacentes são explicitamente ordenadas em cada nó antes de serem combinadas, de maneira consistente entre as partições de malha (por exemplo, ordem crescente de valor). Isso requer que a contribuição de um elemento seja temporariamente armazenada ou a computação replicada para cada nó adjacente a ele. Na segunda, uma ordem consistente é imposta à visitação dos elementos adjacentes a cada nó. Dessa forma, as contribuições dos elementos são naturalmente combinadas de maneira simétrica em qualquer partição.

Conforme discutido na Seção 4.3, uma computação sequencial do tipo *elementos-a-nó* é tradicionalmente realizada percorrendo-se os elementos da malha de elementos finitos e adicionando-se a contribuição de cada elemento aos nós adjacentes. Como consequência, evita-se a replicação de computações nos elementos para os nós adjacentes e a necessidade de armazenamento de dados temporários. De forma que a computação ocorra de maneira simétrica, os elementos devem ser visitados segundo uma ordem global consistente em relação à malha distribuída. Para isso, propõe-se uma abordagem baseada na

ordenação global *implícita* de elementos. A fim de que o percorrimento ordenado seja feito de maneira transparente à aplicação cliente, *iteradores estáveis* de elementos são adicionados à representação de malha distribuída de ParTopS.

A implementação de um iterador estável é baseada em vetores ordenados de identificadores de elementos (*owner_partition, owner_element_handle*). Cada partição de malha mantém um vetor independente, que é atualizado (reordenado) sempre que elementos são adicionados ou removidos dela. O vetor contém apenas elementos do tipo *proxy*, uma vez que os elementos locais possuem a mesma partição proprietária (a partição corrente), e, assim, já se encontram implicitamente ordenados com respeito a ela. A iteração ordenada é feita da seguinte forma: visitam-se os elementos do vetor ordenado, enquanto o identificador da partição proprietária correspondente (*owner_partition*) for menor que o da partição corrente. Então, os elementos locais são visitados, segundo a ordenação local implícita da partição, e o percorrimento do vetor é retomado a partir da posição seguinte à última visitada.

A ordenação implícita de elementos é obtida a partir da comparação lexicográfica dos pares ordenados (*owner_partition, owner_element_handle*), que identificam univocamente os elementos da malha distribuída (Seção 3.1.1), primeiramente pela partição proprietária (*owner_partition*), em seguida pelo identificador local do elemento naquela partição (*owner_element_handle*). Dessa forma, não é preciso atribuir identificadores globais explícitos a elementos, e os custos de manutenção correspondentes são evitados. Além disso, não se requer comunicação entre partições para a atualização da ordenação global quando a malha é modificada. Essa abordagem difere de outras (Dooley et al., 2009), onde operações de 128 bits de precisão são usadas para garantir a consistência de operações de precisão dupla (64 bits) em nós compartilhados por partições diferentes.

Diferentes vetores ordenados e iteradores podem ser criados para cada grupo distinto de elementos. Nas simulações de fraturas realizadas foram utilizados iteradores distintos para elementos volumétricos e coesivos. Nesse caso, o vetor de elementos volumétricos não precisa ser atualizado quando elementos coesivos são inseridos na malha. Considerando-se o número de elementos coesivos normalmente muito menor que o de volumétricos, assim como o de entidades *proxies* em relação às locais, o esforço de atualização do vetor ordenado durante a simulação de fraturas não é significativo.

Com base na representação de malha distribuída de ParTopS, computações simétricas ocorrem tanto para entidades locais como *proxies*.

Entidadades do tipo *ghost*, que dependem de informações de outras partições, são atualizadas a partir de resultados computados nas entidades locais ou *proxies* remotas correspondentes.

4.5. Interface de programação paralela

A interface de programação (*API - Application Programming Interface*) disponibilizada pelo sistema ParTopS a aplicações numéricas baseia-se no paradigma de programação paralela SPMD *(Single Program Multiple Data)* (Foster, 1995). Dessa forma, um único programa é executado concorrentemente para todas as partições da malha distribuída, e em cada partição processa-se uma parte diferente do conjunto de dados do modelo de elementos finitos original. A comunicação entre partições é baseada no envio de mensagens (Foster, 1995), podendo ser integrada com aplicações desenvolvidas com base no padrão MPI (*Message Passing Interface*) (MPI Forum, 2010).

O sistema ParTopS é implementado internamente na linguagem C++ (Stroustrup, 1997), utilizando-se o sistema Charm++ (Kalé & Krishnan, 1993, 1996), uma infra-estrutura orientada a objetos para o desenvolvimento de aplicações paralelas genéricas baseada no envio e recebimento assíncrono de mensagens entre processadores. Esse paradigma oferece oportunidades para a melhoria do desempenho de aplicações paralelas, pois facilita a sobreposição entre etapas de computação e comunicação em operações executadas em paralelo. Por outro lado, também tende a aumentar significativamente a complexidade de programação. Assim, considerando-se a complexidade intrínseca de simulações numéricas, optou-se por prover à aplicação cliente uma interface funcional de programação, mais simples e fácil de usar. O objetivo é facilitar a paralelização de aplicações numéricas existentes, e permitir ao desenvolvedor da aplicação se concentrar no problema numérico.

A interface de programação para aplicações numéricas consiste de funções locais, na linguagem C (Kerninghan & Ritchie, 1988), para o acesso e manipulação de dados em cada partição de malha, mais um conjunto reduzido de *funções paralelas coletivas* (executadas concorrentemente por todas as partições) para as operações envolvendo partições diferentes. Com isso, pretende-se manter o sistema tão simples quanto possível, embora ainda eficiente e escalável a um grande número de processadores. A malha sequencial local de cada partição é representada pela estrutura de dados topológica TopS,

estendida com a infra-estrutura de comunicação entre partições de ParTopS (camada de comunicação, inserção de elementos coesivos em paralelo, iteradores estáveis de elementos). As funções locais proveem o acesso à representação de malha. As funções paralelas coletivas encapsulam a comunicação entre partições diferentes.

4.5.1. Funções exportadas à aplicação numérica

Uma breve descrição das principais funções da interface de programação paralela fornecida à aplicação numérica é apresentada abaixo:

```
void topParInit();
TopParModel topParModel_Create();
int topParModel_ReadMesh(TopParModel model,
    char* format, char* meshfile, char* partfile);
TopModel* topParModel_GetLocalMesh(
    TopParModel  model);
TopAttribId topParModel_CreateNodeDenseAttrib(
    TopParModel model, size_t sizeof_attrib);
void* topModel_GetNodeDenseAttribAt(
    TopModel* mesh, TopAttribId attribid, TopNode node);
void topParModel_SyncProxyNodeDenseAttrib(
    TopParModel model, TopAttribId attribid);
void topParModel_SyncGhostNodeDenseAttrib(
    TopParModel model, TopAttribId attribid);
void topParModel_SyncFacets(
    TopParModel model, TopFacet* facets);
void topParModel_InsertCohesiveAtFacets(
    TopParModel model, ElemType type, TopFacet* facets);
```

A função coletiva `topParInit()`, chamada concorrentemente por todas as partições, inicializa o suporte à representação de malha distribuída; ela deve ser chamada uma vez ao início do programa, após a inicialização do padrão MPI: `MPI_Init()` (MPI Forum, 2010). A função coletiva `topParModel_Create()` cria um novo modelo de elementos finitos distribuído (inicialmente contém apenas uma malha vazia) e retorna o identificador do

modelo criado (`TopParModel`). A função coletiva utilitária `topParModel_ReadMesh()` lê uma malha distribuída no formato `format`, para um modelo distribuído de elementos finitos (`model`), dado um arquivo contendo a malha (`meshfile`) e outro com a descrição das partições (`partfile`). A função local `topParModel_GetLocalMesh()` retorna um ponteiro para a malha local da partição corrente, representada pela estrutura de dados topológica sequencial TopS estendida com o suporte paralelo. A função coletiva `topParModel_CreateNodeDenseAttrib()` adiciona ao modelo (`model`) um atributo associado a todos os nós da malha distribuída, de tamanho `sizeof_attrib` por nó. O identificador do atributo criado (`TopAttribId`) é retornado pela função. Esse identificador é idêntico em todas as partições de malha e pode ser usado para acessar o atributo correspondente a um nó de uma partição qualquer, através de chamada à função local `topModel_GetNodeDenseAttribAt()`, que retorna o ponteiro (`void*`) para a posição de memória correspondente aos dados de um nó (`node`) da malha local (`mesh`). As funções `topParModel_SyncProxyNodeDenseAttrib()` e `topParModel_SyncProxyNodeDenseAttrib()` sincronizam nós *proxies* e *ghosts*, respectivamente, em relação a um atributo (`attribid`) associado aos nós reais remotos correspondentes. A função coletiva `topParModel_SyncFacets()` sincroniza as listas locais de facetas fraturadas (`facets`) de todas as partições de malha com as partições vizinhas. Finalmente, a função coletiva `topParModel_InsertCohesiveAtFacets()` insere adaptativamente elementos coesivos de um determinado tipo (`type`) em um conjunto de facetas fraturadas (`facets`), que inclui consistentemente facetas locais e *proxies* da partição corrente. A aplicação é notificada sobre os elementos coesivos criados e os nós duplicados durante a inserção dos elementos através de funções de chamada de retorno (*callback functions*) registradas na malha local de cada partição pela aplicação numérica.

4.5.2. Implementação da interface de programação

A interface funcional de programação é construída com base na biblioteca TCharm (*Threaded Charm++*) (University of Illinois, 2011), disponibilizada pelo sistema Charm++, que permite o mapeamento do paradigma de objetos e mensagens assíncronas em uma visão baseada em linhas de execução

(*threads*), convencionalmente empregada em aplicações paralelas. Dessa forma, o código numérico sequencial é executado por cada partição de malha em uma linha de execução própria. Com base em TCharm, chamadas a funções paralelas definidas pelo padrão MPI são mapeadas, de forma transparente à aplicação numérica, à infra-estrutura de comunicação assíncrona de Charm++, através da biblioteca AMPI (Huang et al., 2003). As linhas de execução criadas com TCharm podem ser migradas entre processadores e utilizar facilidades de balanceamento de carga oferecidos por Charm++.

No paradigma de programação de Charm++, dois objetos paralelos (C*hares*), *P1* e *P2*, localizados em processadores diferentes, comunicam-se através de chamadas remotas de métodos assíncronas. No exemplo ilustrado na Figura 34, o método *a()*, executando em *P1*, envia uma chamada assíncrona ao método *compute(),* de *P2,* e continua a execução de *a()*. A chamada a *compute()* inclui uma mensagem (*msg*) contendo os parâmetros do método mais uma referência a um método de retorno (*cb()*) em *P1*. Ao terminar a execução de *b()*, o objeto *P2* torna-se disponível ao processamento de novas mensagens (a execução de métodos em cada processador não é interrompida pelo recebimento de mensagens – execução *não preemptiva*). Então, a chamada a *compute()*, recebida e armazenada em uma fila de espera, é processada e o método executado. Ao final da execução de *compute()*, o método de retorno *cb()* é chamado, de maneira assíncrona, para informar *P1* sobre o término de *compute()* e permitir que as computações subsequentes em *P1* sejam executadas. O encadeamento de computações dentro de um objeto também é feito através de chamadas assíncronas, enviadas a si próprio. Na Figura 34, o método *a()*, ao terminar sua execução em *P1*, chama o método *c()* do mesmo objeto; a chamada é armazenada na fila de espera e processada após as qu estiverem pendentes. A implementação interna dos operadores topológicos de ParTopS é realizada com base nesse paradigma.

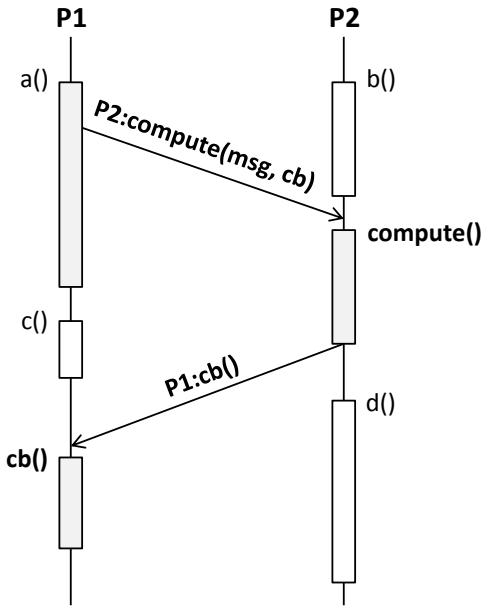

Figura 34 - Comunicação por chamadas remotas de métodos assíncronas entre dois objetos em processadores diferentes (P1 e P2).

Para encapsular as chamadas de métodos assíncronas em um paradigma funcional, criou-se um objeto de contexto associado a uma linha de execução de TCharm. Esse é um objeto regular (*Chare*) de Charm++, contendo referências à linha de execução da aplicação e ao objeto (*Chare*) correspondente à partição de malha. Cada função da interface de programação exportada para a aplicação cliente chama um método correspondente do objeto de contexto. O método do contexto, por sua vez, chama o método da partição de malha, passando como parâmetro a referência a outro método do contexto que deve ser chamado como retorno do método da partição. Então, requisita a suspensão da linha de execução (*thread*) da aplicação. Quando o método de retorno (*callback*) é chamado, indicando a finalização da operação paralela, este requisita a continuação da linha de execução (*thread*) da aplicação. Dessa forma, a função exportada é vista pela aplicação de maneira síncrona, da mesma forma que funções sequenciais regulares. Um exemplo relativo à sincronização de facetas fraturadas é mostrado abaixo, em pseudocódigo (estilo linguagem C++ (Stroustrup, 1997)):

```
// Objeto de contexto
Context {
  TCharm* thread;
  TopPartition* partition;

  void SyncFacets(TopFacet* facets) {
    Callback callback(Context::RecvSyncFacetsReply);
    partition->SyncFacets(facets, callback);
    thread->suspend();
  }

  void RecvSyncFacetsReply ()  {
    thread->resume();
  }
}

// Função exportada à aplicação numérica
void topParModel_SyncFacets (TopFacet* facets) {
  context->SyncFacets(facets);
}
```


4.6. Estrutura da simulação paralela

A estrutura da simulação paralela é determinada a partir dos requisitos de sincronização e padrões de computação discutidos nas seções anteriores. De forma geral, o código paralelo corresponde ao sequencial original, executado concorrentemente por cada partição de malha e com funções paralelas coletivas adicionais para sincronização de atributos de simulação entre partições. A aplicação numérica paralela é apresentada a seguir, considerando-se as abordagens para sincronização de atributos entre partições introduzidas na Seção 4.1. A abordagem empregada define os requisitos de sincronização da aplicação.

4.6.1. Abordagem baseada em computações apenas em entidades locais

Na estrutura da simulação paralela ilustrada na Figura 35, os resultados de computações numéricas são armazenados apenas em entidades locais de cada partição. Isso corresponde à primeira classe de abordagens paralelas da Seção 4.1, sendo equivalente à estratégia convencional de uso de travas de acesso exclusivo (*locks*) – no caso, o *lock* é implícito às entidades locais. Entidades da camada de comunicação (*proxies* e *ghosts*) são consideradas não editáveis e, assim, servem unicamente ao propósito de fornecer os dados necessários a computações locais próximas às fronteiras originais da partição. A consistência de dados de entidades *proxies* e *ghosts* é garantida através de sincronizações com as partições vizinhas, realizadas sempre que necessário. As etapas de sincronização de atributos numéricos são destacadas em negrito na Figura 35.

```
Inicialize o modelo de elementos finitos

Para cada passo de tempo
  1 - Compute o valor das condições de contorno
  2 - Verifique a inserção de novos elementos coesivos
    2.1 - Compute as tensões nodais
    A1. Sincronize atributos de nós proxies
    2.2 - Determine as facetas fraturadas
    A2. Sincronize o conjunto de facetas fraturadas
    2.3 - Insira elementos coesivos
    2.4 - Atualize os valores das massas nodais
  3 - Compute o vetor de forças nodais internas
  4 - Compute o vetor de forças coesivas
  A3. Sincronize atributos de elementos coesivos proxies
  5 - Atualize as acelerações e velocidades nodais
  6 - Atualize as forças externas e energias nos nós
  7 - Aplique as condições de contorno aos nós
  8 - Atualize os deslocamentos nodais
  A4. Sincronize atributos de nós proxies
  A5. Sincronize atributos de nós ghosts
```

Figura 35 - Estrutura do fluxo de controle da simulação de fraturas Modo I, segundo a abordagem baseada em computações locais.

Ao início de um passo de tempo da simulação, assume-se que todas as partições do modelo de elementos finitos encontram-se consistentes entre si. Durante a execução do passo, entretanto, os atributos de nós e elementos são modificados por computações intermediárias, realizadas independentemente em

cada partição. Esses atributos devem estar consistentes para que possam ser utilizados em computações subsequentes. Os requisitos de sincronização para a manutenção da consistência de dados entre partições são analisados a seguir, com base nos padrões de computação observados na estrutura da simulação numérica. A discussão é baseada no detalhamento da simulação numérica sequencial apresentado na Seção 4.2.

A computação de tensões nodais (linha 2.1) consiste em duas etapas sequenciais. Primeiramente, para cada elemento volumétrico, os atributos dos nós adjacentes são trazidos temporariamente ao elemento, para a integração das tensões; em seguida, a contribuição do elemento é extrapolada para ser armazenada nos nós *locais* adjacentes (note que não é necessário armazenar os resultados temporários de elementos). Um nó local pode ser adjacente a elementos locais ou *proxies*; assim as contribuições desses elementos devem ser determinadas. A contribuição de um elemento, por sua vez, depende dos nós adjacentes a ele. Como um elemento *proxy* pode ser adjacente a nós locais, *proxies* ou *ghosts*, os atributos dessas entidades devem estar consistentes antes do início da computação de tensões. Isso é garantido pela sincronização de nós *proxies* e *ghosts* realizada ao final do passo anterior (linhas A4 e A5), após a computação dos deslocamentos nodais.

A determinação de fraturas (linha 2.2) é realizada de forma consistente para todas as facetas *locais* da partição corrente. A inserção de elementos coesivos (Seção 3.2), entretanto, é realizada por meio de uma função paralela coletiva com entrada e saída definidas. Essa função requer que tanto facetas fraturadas locais quanto *proxies* sejam fornecidas. Dessa forma, o conjunto local de facetas fraturadas da partição é sincronizado com as partições vizinhas (linha A2). Uma vez que uma faceta local pode ser adjacente tanto a nós locais quanto *proxies*, e os atributos dos nós remotos correspondentes foram alterados pela computação de tensões, os nós *proxies* devem ser sincronizados antes da determinação de fraturas (linha A1). Após a inserção de elementos coesivos (linha 2.3), a topologia global da malha distribuída e os atributos correspondentes às entidades modificadas encontram-se consistentes.

As massas nodais (linha 2.4) são atualizadas com base nas contribuições de elementos volumétricos adjacentes. Como a massa de um elemento não se altera durante a simulação, e computações subsequentes não requerem massas de nós *proxies* ou *ghosts*, a sincronização de atributos nodais não é necessária antes ou após a computação.

O vetor de forças internas (linha 3) é calculado em duas etapas, de maneira similar às tensões nodais. Para cada elemento volumétrico, atributos de nós adjacentes são trazidos temporariamente ao elemento e combinados com outros atributos estáticos do elemento. Então, a contribuição do elemento é adicionada aos nós *locais* adjacentes a ele. Como o único atributo nodal utilizado nessa computação corresponde ao deslocamento, que não foi alterado até o momento no passo corrente, a sincronização de entidades *proxies* e *ghosts* não é necessária para a determinação das forças internas.

A computação de forças coesivas (linha 4) pode ser decomposta em três partes. Na primeira, para cada elemento coesivo, atributos temporários (separações coesivas) são calculados no elemento a partir de atributos (deslocamentos) de nós adjacentes. Na segunda, atributos do elemento (ex. materiais e história de carregamento) são atualizados, combinando-os com os atributos temporários anteriores, e os resultados armazenados novamente no elemento, se ele for do tipo *local*. Finalmente, a contribuição de forças do elemento é adicionada a cada nó local adjacente. A computação modifica tanto elementos coesivos locais quanto nós locais adjacentes a eles. Uma vez que nós locais podem ser adjacentes a elementos locais e *proxies*, e esses elementos a nós locais, *proxies* e *ghosts*, a computação depende dos atributos dessas entidades. Porém, considerando que os atributos nodais utilizados não foram alterados no passo corrente, é preciso sincronizar apenas atributos de elementos coesivos (linha A3), para que estejam consistentes ao passo seguinte.

As computações restantes (acelerações e velocidades, forças externas e energias, condições de contorno e deslocamentos nodais) (linhas 5 a 8) dependem apenas de atributos dos próprios nós que modificam. Dessa forma, as computações são realizadas para cada nó local da partição corrente e não dependem de sincronizações de entidades. Conforme discutido anteriormente, ao final do passo de simulação, os atributos de entidades *proxies* e *ghosts* são sincronizados (linhas A4 e A5), para que o estado da simulação esteja consistente para o passo seguinte.

Com a edição de atributos de um elemento ou nó realizada apenas pela partição proprietária da entidade, divergências numéricas entre partições da malha distribuída são naturalmente evitadas. Por outro lado, atributos de entidades *proxies* e *ghosts* devem ser sincronizados de maneira apropriada.

4.6.2. Abordagem baseada em computações replicadas

Nesta abordagem, a replicação de computações numéricas em elementos e nós da camada de comunicação é explorada a fim de se reduzir a sincronização de atributos necessária à manutenção da consistência de dados entre partições de malha. Assim, as entidades da camada de comunicação são consideradas editáveis, e resultados de computações armazenados nelas sempre que conveniente.

A seguir, a estrutura da simulação paralela da Seção 4.6.1 é revisitada (Figura 36), e os padrões de computação que nela ocorrem são analisados. Os resultados de computações replicadas são tratados como idênticos em todas as partições, sem modificações adicionais à simulação original para garantir uma ordem consistente de execução de operações de ponto flutuante. Em computações do tipo *elementos-a-nó*, isso pode levar a pequenas diferenças de resultados entre partições. Para limitar a possível propagação de diferenças ao longo de passos de tempo da simulação, realizam-se sincronizações esporádicas dos atributos numéricos das entidades topológicas afetadas.

```
Inicialize o modelo de elementos finitos

Para cada passo de tempo
  1 - Compute o valor das condições de contorno
  2 - Verifique a inserção de novos elementos coesivos
    2.1 - Compute as tensões nodais
    Se o passo de tempo é múltiplo de n
      A1. Sincronize atributos de nós proxies
    2.2 - Determine as facetas fraturadas
    A2. Sincronize o conjunto de facetas fraturadas
    2.3 - Insira elementos coesivos
    2.4 - Atualize os valores das massas nodais
  3 - Compute o vetor de forças nodais internas
  4 - Compute o vetor de forças coesivas
  Se o passo de tempo é múltiplo de n
    A3. Sincronize atributos de nós proxies
  A4. Sincronize atributos de nós ghosts
  5 - Atualize as acelerações e velocidades nodais
  6 - Atualize as forças externas e energias nos nós
  7 - Aplique as condições de contorno aos nós
  8 - Atualize os deslocamentos nodais
```

Figura 36 - Estrutura da simulação numérica paralela baseada em computações replicadas. A sincronização de nós *proxies* ocorre esporadicamente, em intervalos de *n* passos de tempo.

Na etapa inicial de cada passo de tempo da simulação, as tensões nodais são computadas (linha 2.1). Para isso, os valores temporários de um elemento volumétrico são calculados a partir dos atributos dos nós adjacentes, o que corresponde ao padrão *nós-a-elemento*. Assim, os resultados obtidos no elemento são naturalmente simétricos, se apresentando consistente entre partições diferentes. Na segunda etapa, entretanto, a contribuição do elemento é adicionada aos nós adjacentes, segundo o padrão *elementos-a-nó*. Nesse caso, a aplicação calcula os resultados nodais da forma tradicional, percorrendo os elementos da partição corrente, e adicionando a contribuição de cada elemento aos nós adjacentes.

Assumindo-se que nenhuma ordenação global é imposta ao percorrimento de elementos, resultados numéricos ligeiramente divergentes para um mesmo nó podem ser obtidos por partições diferentes, o que requer a sincronização esporádica de nós *proxies* para se manter a consistência dos dados (linha A1). Se a sincronização fosse completamente eliminada, as divergências numéricas se propagariam para passos de tempo subsequentes, o que poderia afetar a convergência correta da simulação. Por outro lado, divergências que não alterem significativamente a simulação poderiam ser ignoradas (ou toleradas). Assim, como forma de se remover o custo de sincronização, mas ao mesmo tempo mitigar o acúmulo de divergências numéricas, a sincronização de *proxies* é realizada em intervalos determinados de *n* passos de tempo. Nesta pesquisa, executaram-se simulações numéricas com sincronização de nós *proxies* em intervalos de 100 passos, com base na aplicação numérica Modo I considerada como exemplo. Nenhuma diferença significativa foi observada quanto aos resultados obtidos.

A determinação de facetas (linha 2.2) pode ser representada pelo padrão de computação *nós-a-elemento*. Assim, considerando-se que os nós de cada faceta encontram-se consistentes, a computação é naturalmente simétrica. Com isso, a sincronização de conjuntos locais de facetas fraturadas pode ser eliminada se a determinação de fraturas em cada partição incluir também facetas do tipo *proxy*. Entretanto, as divergências numéricas da computação de tensões, mesmo pequenas, podem afetar decisões binárias, como a do critério de fratura, fazendo com que os conjuntos de facetas fraturadas fiquem inconsistentes entre as partições. Dessa forma, para garantir a consistência dos dados, a determinação de facetas permanece sendo feita com base em facetas locais apenas, seguida pela sincronização dos conjuntos de facetas (linha A2).

Assim como a computação de tensões nodais, o vetor de forças internas (linha 3) é computado em uma etapa baseada no padrão *nós-a-elemento*, seguida de outra do tipo *elementos-a-nó*. Por sua vez, a computação do vetor de forças coesivas (linha 4) corresponde aos padrões *nós-a-elemento*, *elemento-a-elemento* e *elementos-a-nó*. As computações *nós-a-elemento* e *elemento-a-elemento* são simétricas e dependem de atributos nodais (ex. deslocamentos e materiais) não alterados no passo corrente. As computações *elemento-a-elemento* alteram e dependem de atributos de elementos coesivos calculados no passo anterior. Porém, como essas computações são também simétricas, os resultados não precisam ser sincronizados para se garantir a consistência do passo seguinte. Finalmente, as computações *elementos-a-nó* se baseiam nos resultados das anteriores, e assim dependem de dados já consistentes. Os resultados são tratados de maneira similar às tensões nodais. Assim, a sincronização de nós *proxies* é também realizada em intervalos de *n* passos de tempo, e apenas atributos de nós *ghosts* devem ser sincronizados antes de serem usados nas computações seguintes, ou ao final do passo de tempo (linhas A3 e A4).

As computações restantes (acelerações e velocidades, forças externas e energias, condições de contorno e deslocamentos nodais) (linhas 5 a 8) seguem o padrão *nó-a-nó*. Logo, ocorrem de forma simétrica para todos os nós.

4.6.3. Abordagem baseada em computações simétricas

A sincronização esporádica de nós *proxies* da abordagem anterior pode ser eliminada de forma transparente à aplicação numérica através do uso de iteradores estáveis (Seção 4.4.1) para o percorrimento de elementos em computações *elementos-a-nó*. Nesse caso, os iteradores da aplicação original são diretamente substituídos pelas versões estáveis correspondentes. Com consequência, resultados de tensões nodais e forças internas e coesivas, por exemplo, passam a ser computados de maneira simétrica, produzindo resultados numericamente consistentes para nós locais e *proxies* de cada partição, sem a necessidade de sincronização. Com a determinação de fraturas realizada também para facetas *proxies*, além das locais, a sincronização de conjuntos de facetas é removida. Por outro lado, facetas *proxies* podem ser adjacentes a nós *ghosts*, e, assim, esses nós devem ser sincronizados antes da determinação de facetas. O código paralelo baseado nessa estratégia é ilustrado na Figura 37, e

requer apenas sincronização de nós *ghosts* (linhas A1 e A2). Uma vantagem da redução de pontos de sincronização através de computações estáveis é a simplificação significativa da estrutura da simulação paralela.

```
Inicialize o modelo de elementos finitos

Para cada passo de tempo
  1 - Compute o valor das condições de contorno
  2 - Verifique a inserção de novos elementos coesivos
    2.1 - Compute as tensões nodais
    A1. Sincronize atributos de nós ghosts
    2.2 - Determine as facetas fraturadas
    2.3 - Insira elementos coesivos
    2.4 - Atualize os valores das massas nodais
  3 - Compute o vetor de forças nodais internas
  4 - Compute o vetor de forças coesivas
  A2. Sincronize atributos de nós ghosts
  5 - Atualize as acelerações e velocidades nodais
  6 - Atualize as forças externas e energias nos nós
  7 - Aplique as condições de contorno aos nós
  8 - Atualize os deslocamentos nodais
```

Figura 37 - Estrutura da simulação paralela utilizando iteradores estáveis de elementos.

4.6.4. Abordagem mista

Com o emprego de computações simétricas, os pontos de sincronização de atributos da simulação numérica são reduzidos em relação à abordagem convencional baseada apenas em computações locais. Contudo, dependendo da aplicação, a combinação de abordagens diferentes pode se tornar mais vantajosa.

No caso da simulação numérica analisada, a sincronização de conjuntos de facetas fraturadas não é necessária quando computações simétricas são empregadas. Por outro lado, isso requer que atributos de nós *ghosts* sejam sincronizados antes da determinação de fraturas. Porém, um volume menor de dados é, em geral, esperado na sincronização de conjuntos de facetas em comparação com atributos de nós *ghosts*. Assim, uma abordagem mais compacta utiliza computações simétricas, mas mantém a determinação de fraturas a partir de facetas locais, com a sincronização dos conjuntos de facetas correspondentes. A estrutura da simulação paralela resultante é apresentada na Figura 38.

```
Inicialize o modelo de elementos finitos

Para cada passo de tempo
  1 - Compute o valor das condições de contorno
  2 - Verifique a inserção de novos elementos coesivos
    2.1 - Compute as tensões nodais
    2.2 - Determine as facetas fraturadas
    A1. Sincronize o conjunto de facetas fraturadas
    2.3 - Insira elementos coesivos
    2.5 - Atualize os valores das massas nodais
  3 - Compute o vetor de forças nodais internas
  4 - Compute o vetor de forças coesivas
  5 - Atualize as acelerações e velocidades nodais
  6 - Atualize as forças externas e energias nos nós
  7 - Aplique as condições de contorno aos nós
  8 - Atualize os deslocamentos nodais
  A2. Sincronize os atributos de nós ghosts
```

Figura 38 - Estrutura da simulação paralela segundo a abordagem mista, com iteradores estáveis e sincronização dos conjuntos de facetas fraturadas.