Simulação Distribuída e Paralela de Mobilidade Territorial Sazonal

Projeto de Implementação

Introdução ao Processamento Paralelo e Distribuído

2025/2

1 Contextualização

Diversos povos indígenas brasileiros organizam o uso do território de forma distribuída, sazonal e não centralizada. Grupos familiares se deslocam ao longo do território conforme ciclos ambientais, disponibilidade de recursos e restrições culturais, sem a existência de uma autoridade central que coordene essas decisões.

Do ponto de vista computacional, essa realidade pode ser modelada como um sistema multiagente espacial de grande escala, no qual:

- milhares de agentes tomam decisões locais simultaneamente;
- o ambiente evolui continuamente no tempo;
- há interação indireta entre agentes por meio do território;
- a simulação exige escalabilidade computacional.

Esse tipo de problema demanda:

- distribuição do domínio entre processos (MPI);
- paralelismo intra-processo para atualização massiva de agentes e células (OpenMP).

Portanto, o problema é naturalmente adequado a um modelo de **computação híbrida MPI + OpenMP**.

2 Definição do problema

O objetivo é implementar uma simulação distribuída e paralela de um território modelado como um grid 2D, no qual grupos familiares indígenas se deslocam sazonalmente em busca de recursos.

2.1 Território

O território é representado como uma grade bidimensional de células. Cada célula possui os seguintes atributos:

- tipo: aldeia, pesca, coleta, roçado ou área interditada;
- recurso: valor numérico escalar que representa a capacidade local de sustentar agentes durante um ciclo;
- acessibilidade, dependente da estação (seca ou cheia).

O recurso é um valor abstrato e finito, associado a cada célula, que pode ser interpretado como disponibilidade de alimento, capacidade produtiva ou suporte ambiental. Esse valor é consumido pelos agentes ao longo da simulação e pode regenerar parcialmente de acordo com a estação vigente.

O grid global é particionado em blocos retangulares contíguos de células, sendo cada bloco atribuído a um processo MPI como seu subgrid local.

Na inicialização do grid, inicializar o gerador de números aleatórios com uma semente fixa, para garantir reprodutibilidade.

2.2 Agentes

Cada agente representa um grupo familiar e possui:

- posição no grid;
- estado interno (necessidade ou energia);
- regras locais de decisão para deslocamento.

Os agentes:

- avaliam regiões vizinhas;
- decidem deslocamento ou permanência;
- consomem parte do recurso da célula ocupada.

O consumo de recurso envolve a execução de uma carga computacional sintética, parametrizada pelo valor do recurso local, representando o custo de avaliação e tomada de decisão do agente. Essa carga não modela um fenômeno físico real, sendo utilizada exclusivamente para controlar o volume de computação e justificar o paralelismo intra-processo.

2.3 Tempo e sazonalidade

A simulação evolui em ciclos discretos. Um conjunto fixo de ciclos representa uma estação. A mudança de estação altera:

- acessibilidade das regiões;
- taxas de regeneração dos recursos.

2.4 Carga computacional sintética

A carga computacional sintética representa o custo associado ao consumo de recursos e à tomada de decisão dos agentes. Ela é implementada como um trabalho computacional parametrizável, proporcional ao valor do recurso disponível na célula ocupada pelo agente.

Essa carga pode ser implementada, por exemplo, como um laço de iterações cujo número depende do recurso local, realizando operações aritméticas simples. O objetivo não é simular um processo físico específico, mas controlar o custo computacional por agente, permitindo a análise de desempenho e escalabilidade da solução híbrida MPI + OpenMP.

Importante: O custo computacional por agente deve ser limitado por um valor máximo pré-definido, de modo a evitar cargas excessivas que inviabilizem a execução da simulação.

3 Modelo de paralelismo

- MPI é utilizado para distribuir espacialmente o território entre processos.
- OpenMP é utilizado para paralelizar, dentro de cada processo:
  - o processamento massivo de agentes;
  - a atualização das células do grid local.

O uso de OpenMP é considerado **necessário**, não opcional, devido ao alto número de agentes por partição e ao custo computacional das decisões locais.

O território global é dividido em blocos espaciais contíguos de células (subgrids), e cada bloco é atribuído a um processo MPI.

4 Algoritmo base da simulação

Atenção: Considere que as dimensões do grid global são múltiplas do número de processos MPI, de modo a simplificar a divisão do domínio.

```
Algoritmo SimulacaoHibrida_MPI_OpenMP

Entrada:
  W, H        dimensões globais do grid
  T           número total de ciclos
  S           tamanho do ciclo sazonal
  N_agents    número total de agentes

1) Inicializar MPI
   MPI_Init()
   rank <- MPI_Comm_rank()
   P    <- MPI_Comm_size()

2) Particionar o grid global
   Cada processo recebe um subgrid local
   Definir offsets (offsetX, offsetY)

3) Inicializar grid local (determinístico)
   Para cada célula local (i,j):
     gx <- offsetX + i
     gy <- offsetY + j
     tipo <- f_tipo(gx, gy)
     recurso <- f_recurso(tipo)
     acessivel <- f_acesso(tipo, estacao_inicial)

4) Inicializar agentes locais
   Criar agentes apenas em regiões do subgrid local

5) Para t = 0 até T-1:

   5.1) Atualizar estação (se aplicável)
        Se (t % S == 0):
          estacao <- alternar(estacao)
          MPI_Bcast(estacao)

   5.2) Troca de halo do grid (MPI)
        Enviar e receber bordas do subgrid

```

```
   5.3) Processar agentes (OpenMP)
        #pragma omp parallel
          buffers_envio_por_thread <- vazio
          lista_local_thread <- vazio

          #pragma omp for
          Para cada agente a:
            celula_atual <- posicao(a)
            r <- recurso(celula_atual)

            // Carga sintética proporcional ao recurso
            executar_carga(r)

            destino <- decidir(a, grid_local, halo)

            Se destino é local:
              consumir_recurso(celula_atual, a)
              adicionar a em lista_local_thread
            Senão:
              identificar processo destino
              adicionar a em buffer_envio_por_thread
    Onde:
        executar_carga(r) representa um trabalho computacional sintético proporcional
        ↪ ao recurso r, por exemplo um laço de iterações com custo controlado pelo
        ↪ parâmetro r, algo como: for( c = CUSTO ; c >= 0 ; --c ) do_work();

   5.4) Migrar agentes (MPI)
        Enviar buffers de agentes para processos vizinhos
        Receber agentes migrados
        Atualizar lista local de agentes

   5.5) Atualizar grid local (OpenMP)
        #pragma omp parallel for collapse(2)
        Para cada célula local:
          recurso <- recurso
                     + regeneracao(estacao)
                     - consumo_acumulado_na_celula

   5.6) Métricas globais (MPI)
        MPI_Allreduce(medidas locais)

   5.7) Sincronização opcional
        MPI_Barrier()

6) Finalizar MPI
   MPI_Finalize()
```

5 Observações

- A migração de agentes caracteriza a natureza distribuída do problema.
- O paralelismo OpenMP é essencial para desempenho intra-nó.
- Implementações que não explorem efetivamente OpenMP serão consideradas incompletas.