Universidade Federal do Rio Grande do Norte - UFRN
Departamento de Engenharia da Computação e Automação - DCA
DCA3703 - Programação Paralela

Tarefa 07: Utilização de tasks

Discente: Quelita Míriam
Docente: Samuel Xavier de Souza

I. INTRODUÇÃO

Neste relatório, explora-se os conceitos fundamentais da programação paralela com a biblioteca OpenMP, com ênfase na criação de tarefas com *#pragma omp task*, e no uso dos *constructs #pragma omp single* e *#pragma omp master*. Discutindo também as implicações das barreiras implícitas e explícitas no controle de fluxo de programas paralelos. Através de uma implementação prática, é demonstrado a execução concorrente de tarefas sobre uma estrutura de dados do tipo lista encadeada.

II. METODOLOGIA

Foi desenvolvido um programa[1] em linguagem C, utilizando a API OpenMP, que simula o processamento paralelo de arquivos representados em uma lista encadeada. Cada nó da lista contém o nome de um arquivo fictício. O processamento consiste em imprimir o nome do arquivo e a thread responsável pela execução da tarefa. O código é compilado a partir de "*gcc main.c -o main -fopenmp -lm && ./main*".

Após a criação de uma lista encadeada com nós, há uma região paralela com #pragma omp parallel. Dentro dessa região, foi utilizado #pragma omp single e #pragma omp master para analisar o comportamento dos constructs com as threads. A partir disso foram feitas observações sobre o comportamento do código e levantadas reflexões sobre determinismo, concorrência e segurança de dados em memória compartilhada.

III. RESULTADOS E DISCUSSÕES

Usar *#pragma omp task* sem *#pragma omp single* ou *#pragma omp master* pode causar problemas, pois todas as threads vão tentar criar tarefas ao mesmo tempo, o que pode gerar condições de corrida ao acessar a variável compartilhada, criação de tarefas duplicadas, dificuldade de controle e depuração (Imagem 1).

```
Processando arquivo4.txt na thread 3
Processando arquivo2.txt na thread 3
Processando arquivo2.txt na thread 0
Processando arquivo1.txt na thread 2
Processando arquivo3.txt na thread 1
```

Imagem 1: Uso de *#pragma omp task* sem *#pragma omp single* ou *#pragma omp master* no código

[1] Link do código da execução da atividade proposta:
https://github.com/quelita2/programacao-paralela/blob/main/topico01/tarefa07/src/main.c

Ao usar *#pragma omp single* ou *#pragma omp master* sem *#pragma omp task* limita o código para ser executado por apenas uma thread dentro da região paralela.

- ***#pragma omp single***: qualquer thread pode executar o trecho, mas somente uma fará isso. Ao final, há barreira implícita, as outras threads esperam (imagem 2).
- ***#pragma omp master***: somente a thread master (thread 0) executa o trecho. Não há barreira implícita ao final, as outras threads continuam executando (imagem 3).

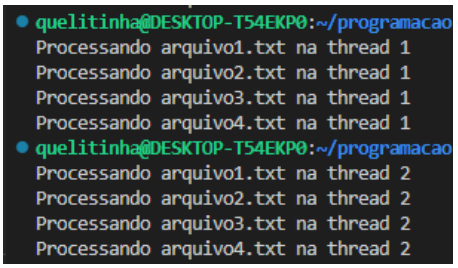

Imagem 2: Uso do *#pragma omp single* sem *#pragma omp task*

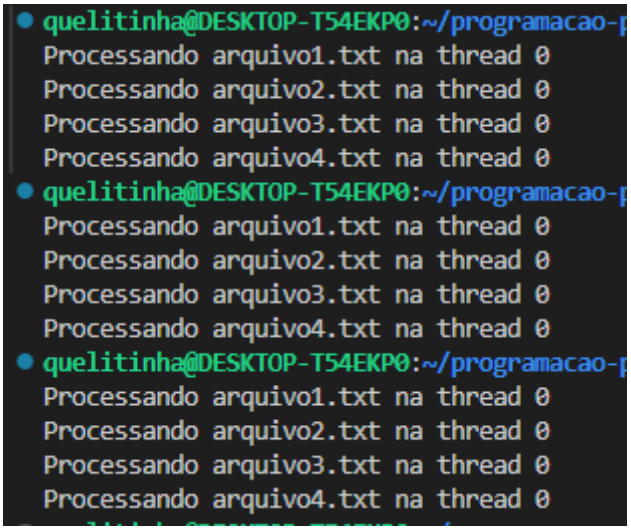

Imagem 3: Uso do *#pragma omp master* sem *#pragma omp task*

Sendo assim, o uso do *#pragma omp task* dentro de um *single* (ou *master*) garante que somente uma thread cria as tarefas e as outras ficam livres para executá-las.

Durante a execução do programa, observou-se que:

A thread que cair no ***#pragma omp single*** percorre a lista e cria as tarefas. As outras threads aguardam até que essa thread termine, por causa da barreira implícita no final do *single*. Ou seja, todas as tarefas são criadas antes que o time de *threads* continue. O comportamento do programa não é determinístico, ou seja, em diferentes execuções, diferentes *threads* podem processar os nós.

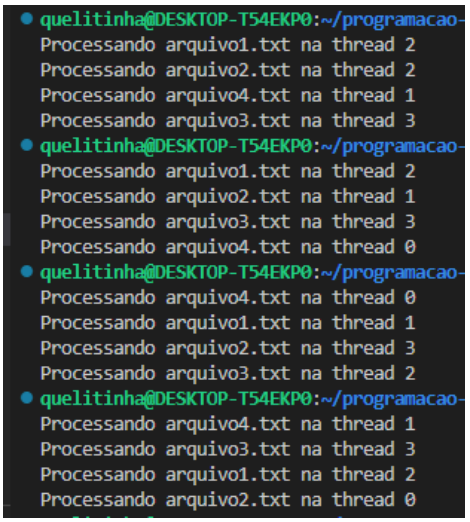

Imagem 4: Comportamento do *#pragma omp single* com *#pragma omp task firstprivate(node)*

Quando não se utiliza ***firstprivate(node)***, pode ocorrer acesso incorreto aos dados, pois todas as tarefas compartilham o mesmo ponteiro *node*, que é atualizado no *loop* externo. Isso pode fazer com que um mesmo nó seja processado múltiplas vezes ou que dados incorretos sejam acessados.

Com o uso do ***#pragma omp master*** somente a *thread* mestre (ID 0) cria as tarefas. Mas como não há barreira implícita, as outras threads não esperam pela criação das *tasks*, o que pode ser problemático se não for usado com ***#pragma omp taskwait*** ou ***#pragma omp barrier***. Contudo, por estar sendo utilizada dentro do ***#pragma omp parallel*****,** possui uma barreira implícita ao final, impede e protege a continuação do programa até que todas as threads completem a região paralela.

IV. CONCLUSÃO

A atividade demonstrou a importância do uso correto das diretivas de OpenMP na programação paralela. A criação de tarefas com *#pragma omp task* permite paralelizar o processamento de forma eficiente, mas requer cuidados com sincronização e escopo de variáveis.

O uso de *#pragma omp single* é recomendado para criar tarefas, pois garante que apenas uma thread execute essa parte do código e impõe uma barreira implícita, evitando que outras threads avancem antes da criação completa das tarefas. Já *#pragma omp master*, por não ter barreira implícita, pode causar problemas se usado sem *#pragma omp barrier*.

Além disso, o uso de *firstprivate(node)* é essencial para que cada tarefa receba sua própria cópia do ponteiro, evitando condições de corrida e acessos incorretos à memória. Sem essa prática, podem ocorrer erros como repetição, perda de tarefas ou falhas de segmentação.

Assim, a correta aplicação dessas diretivas é fundamental para garantir a segurança e eficiência de programas paralelos com OpenMP.