Programação Paralela
Tarefa 7 - Utilização de tasks

Anelma Silva da Costa
UFRN - DCA
anelma.costa.100@ufrn.edu.br

I - Introdução

O avanço da computação paralela tem desempenhado um papel fundamental na melhoria do desempenho de aplicações computacionais que lidam com grandes volumes de dados ou que exigem processamento intensivo. Com a popularização dos processadores multicore, a utilização eficiente dos recursos computacionais tornou-se uma necessidade. Nesse contexto, a linguagem C, amplamente utilizada no desenvolvimento de sistemas e aplicações de baixo nível, oferece suporte à programação paralela por meio da biblioteca OpenMP (Open Multi-Processing), permitindo a criação de regiões paralelas e tarefas para explorar o paralelismo de forma mais intuitiva e estruturada.

Este relatório tem como objetivo descrever a implementação de uma aplicação paralela utilizando C e OpenMP, na qual uma lista encadeada de arquivos fictícios é percorrida dentro de uma região paralela, e cada nó da lista é processado por uma tarefa distinta. A proposta visa explorar os conceitos de paralelismo de tarefas e gerenciamento de dependências, além de verificar o comportamento da execução paralela sob diferentes condições.

Adicionalmente, serão discutidas as possíveis inconsistências relacionadas ao processamento dos nós da lista, como a duplicidade, a omissão e a variação de execução entre diferentes rodadas do programa. Também serão propostas soluções para garantir que cada nó da lista seja processado uma única vez e por apenas uma tarefa, assegurando a corretude e a eficiência da aplicação paralela.

II - Teoria

A programação paralela é uma técnica que permite a execução simultânea de múltiplas operações, com o objetivo de melhorar o desempenho e a eficiência de programas computacionais. Dentre as ferramentas disponíveis para a implementação de programas paralelos em C, destaca-se o OpenMP, uma API amplamente utilizada por sua simplicidade e poder expressivo. O OpenMP baseia-se no uso de diretivas de compilação que indicam ao compilador quais partes do código devem ser executadas paralelamente.

Um dos recursos mais relevantes do OpenMP para o desenvolvimento de programas baseados em tarefas é a diretiva #pragma omp task, que permite a criação de tarefas independentes que podem ser executadas por qualquer thread disponível no pool de threads da aplicação. Tais tarefas são particularmente úteis quando se deseja atribuir unidades lógicas de trabalho, como o processamento de nós de uma lista, a diferentes threads em tempo de execução.

Contudo, a natureza não determinística da execução paralela pode ocasionar variações no comportamento do programa entre diferentes execuções, mesmo que o código-fonte permaneça inalterado. Isso se deve ao fato de que o agendamento das tarefas é realizado dinamicamente, com base na disponibilidade das threads, o que pode levar à execução de tarefas em ordens distintas ou até mesmo à omissão ou duplicação de tarefas, caso não haja controle adequado.

No caso específico da manipulação de listas encadeadas, cuidados adicionais devem ser tomados. Como cada nó depende do acesso ao próximo, a iteração sequencial é natural; no entanto, ao paralelizar o processamento dos nós, é preciso assegurar que cada tarefa opere sobre uma cópia estável e independente do nó corrente. Para tanto, é comum capturar o ponteiro do nó atual em uma variável local antes da criação da tarefa, garantindo que a tarefa referencia o nó correto.

Além disso, o uso de diretivas como #pragma omp single pode ser necessário para evitar que múltiplas threads iniciem a criação de tarefas simultaneamente sobre a mesma estrutura, o que poderia comprometer a integridade do programa. Portanto, o controle adequado do escopo de paralelização e das variáveis utilizadas nas tarefas é essencial para a execução correta e eficiente da aplicação paralela.

III - Aplicação Prática

A aplicação prática proposta consiste na implementação de um programa em linguagem C (Link do código completo: https://github.com/AnelmaSilva/Programa-ao_Paralela_25_1/tree/main/Tarefa%207) que utiliza a API OpenMP para criar tarefas paralelas com o objetivo de processar, individualmente, cada nó de uma lista encadeada contendo nomes de arquivos fictícios. O programa ilustra de forma clara o uso da diretiva #pragma omp task, destacada na fundamentação teórica, e demonstra como ela pode ser aplicada de forma segura e eficiente no processamento de estruturas de dados dinâmicas.

O programa inicia com a definição da estrutura Node, que representa cada elemento da lista encadeada. Cada nó armazena o nome de um arquivo e um ponteiro para o próximo nó. As funções auxiliares createNode, appendNode e freeList são responsáveis, respectivamente, pela criação, inserção e liberação de memória dos nós da lista.

Na função main, é criada uma lista contendo cinco arquivos simulados. Em seguida, inicia-se uma região paralela com #pragma omp parallel, onde múltiplas threads são disponibilizadas para execução concorrente. Contudo, para evitar que todas as threads iterem simultaneamente sobre a lista, a diretiva #pragma omp single nowait é utilizada. Ela assegura que apenas uma thread execute o loop responsável pela criação das tarefas, evitando condições de corrida e sobreposição de tarefas.

Dentro do laço que percorre a lista encadeada, a diretiva #pragma omp task firstprivate(current) cria uma tarefa para cada nó. A cláusula firstprivate é essencial neste contexto, pois garante que cada tarefa acesse uma cópia própria do ponteiro current no momento da sua criação. Caso contrário, todas as tarefas poderiam acessar o mesmo valor de current, levando à duplicação ou omissão no processamento dos nós — um comportamento indesejado e abordado na fundamentação teórica como um risco do não determinismo da execução paralela.

A função processNode é responsável por imprimir o nome do arquivo associado ao nó e o identificador da thread que o processou, demonstrando a efetiva distribuição das tarefas entre múltiplas threads. Ao final da região paralela, a diretiva #pragma omp taskwait assegura que todas as tarefas tenham sido concluídas antes que a execução continue, evitando que a memória da lista seja liberada prematuramente.

A execução do programa evidencia o comportamento dinâmico do agendador de tarefas do OpenMP. A ordem de processamento dos arquivos pode variar entre diferentes execuções, e diferentes threads podem ser responsáveis por diferentes nós em cada rodada. Contudo, graças ao uso correto da diretiva task com firstprivate, garante-se que cada nó seja processado exatamente uma vez, sem sobreposição ou omissão, como previsto teoricamente.

Essa implementação valida os conceitos explorados na fundamentação teórica, principalmente no que tange ao uso de tarefas com estruturas encadeadas, controle de concorrência e garantias de execução única por nó. A correta utilização das diretivas do OpenMP neste cenário demonstra a viabilidade e os cuidados necessários para o desenvolvimento de aplicações paralelas corretas e eficientes.

IV - Teoria x Resultados Práticos

A execução prática do programa implementado demonstrou, de forma clara, a aplicação correta dos conceitos teóricos previamente discutidos. A saída observada revelou que todos os arquivos fictícios adicionados à lista encadeada foram devidamente processados: documento1.txt, imagem.jpg, relatorio.pdf, planilha.xlsx e apresentação.ppt (Figura abaixo) apareceram uma única vez na impressão dos resultados, cada um associado a uma thread distinta. Esse comportamento confirma a eficácia do uso da diretiva #pragma omp task em conjunto com a cláusula firstprivate(current), que foi destacada na fundamentação teórica como

essencial para garantir que cada tarefa trabalhe com uma cópia independente do nó atual.

```
Arquivo: apresentacao.ppt - Processado pela thread 1
Arquivo: planilha.xlsx - Processado pela thread 5
Arquivo: documento1.txt - Processado pela thread 11
Arquivo: relatorio.pdf - Processado pela thread 16
Arquivo: imagem.jpg - Processado pela thread 14
```

Não houve duplicação nem omissão de nós, o que evidencia que a criação das tarefas foi feita de forma segura e controlada. Isso se deve, principalmente, ao encapsulamento da lógica de criação de tarefas dentro de uma região #pragma omp single, o que evita que múltiplas threads percorram a lista simultaneamente. Ao limitar a criação de tarefas a apenas uma thread, eliminam-se conflitos de acesso à estrutura da lista encadeada. A cláusula firstprivate assegura que o valor de current, capturado no momento da criação da tarefa, seja preservado mesmo após o ponteiro original ser alterado no laço de iteração. Assim, cada tarefa é corretamente associada a um nó distinto da lista, cumprindo sua função de forma única e independente.

Outro ponto notável está na ordem de execução das tarefas e na distribuição das threads. A ordem em que os arquivos foram processados e os identificadores das threads variaram em relação à sequência de inserção dos nós na lista. Isso ilustra uma característica central da programação paralela: o agendamento dinâmico e não determinístico das tarefas pelo ambiente de execução do OpenMP. Como discutido anteriormente, a decisão sobre qual thread executa qual tarefa é tomada em tempo de execução, de acordo com a disponibilidade das threads no pool. Portanto, é esperado que execuções distintas do mesmo programa resultem em diferentes combinações de ordem de execução e alocação de threads.

A sincronização ao final da região paralela, feita por meio da diretiva #pragma omp taskwait, garantiu que todas as tarefas fossem concluídas antes que a memória da lista fosse liberada. Esse detalhe, embora muitas vezes negligenciado, é essencial para evitar o acesso à memória já desalocada e garantir a integridade da execução como um todo.

Dessa forma, o comportamento observado na prática está em total conformidade com os princípios apresentados na teoria. A correta utilização das diretivas do OpenMP, aliada à atenção ao escopo das variáveis e ao controle da concorrência, asseguraram um paralelismo eficiente, seguro e determinístico quanto ao número de execuções por tarefa, mesmo que a ordem e a thread responsável possam variar entre execuções.

V - Conclusões

A atividade proposta proporcionou uma oportunidade prática de consolidar conceitos fundamentais da programação paralela com OpenMP, especialmente no que diz respeito ao uso de tarefas (tasks) para processar estruturas de dados dinâmicas, como listas encadeadas. Através da implementação e análise do código desenvolvido, foi possível observar na prática os desafios e as soluções teóricas que envolvem a criação, execução e sincronização de tarefas paralelas.

O experimento demonstrou que, com o uso adequado das diretivas #pragma omp task, #pragma omp single e #pragma omp taskwait, é possível distribuir de forma eficiente e segura o processamento de elementos de uma lista entre múltiplas threads, sem sobrecarga, duplicidade ou perda de dados. A aplicação correta da cláusula firstprivate mostrou-se crucial para preservar a integridade da informação associada a cada tarefa, evitando comportamentos indesejados que poderiam comprometer o resultado da execução paralela.

Além disso, o comportamento não determinístico observado na ordem de execução e alocação de threads reforça a natureza dinâmica dos ambientes paralelos, exigindo do programador o domínio de estratégias para garantir a corretude, mesmo diante de resultados que variam a cada execução.

Em síntese, a atividade cumpriu com êxito seus objetivos pedagógicos, ao permitir a experimentação concreta dos mecanismos de paralelismo por tarefas, reforçando a importância da compreensão teórica como base para a construção de soluções robustas, corretas e escaláveis em programação paralela. A correlação entre teoria e prática revelou-se essencial para o desenvolvimento de uma visão crítica e aplicada da computação concorrente.

VI - Referências

- Livro: An introduction to parallel programming .
- Conteúdo ministrado em sala de aula.
- IA’s (DeepSeek e ChatGPT)