Exercício de Programação 1 – Fundamentos de Sistemas Paralelos e Distribuídos

Vitor Rodarte Ricoy - 2019007112

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

1. Introdução

O trabalho proposto tem como objetivo treinar e fixar os conceitos aprendidos em aula relativos ao uso da biblioteca pthreads, além de conceitos de paralelização e sincronização. Esse trabalho consiste na implementação de um programa que possui uma thread leitora, responsável por ler tarefas de um arquivo e popular uma fila com essas tarefas, e de diversas threads trabalhadoras, responsáveis por consumir tarefas da fila e executá-las. Por fim, o programa também exibe estatísticas acerca da execução das tarefas pelas threads trabalhadoras.

2. Descrição Geral

Primeiramente o programa checa os argumentos recebidos, para verificar se apenas o nome do arquivo de entrada foi recebido. Após isso, o programa abre o arquivo para leitura e inicializa a thread leitora e, em seguida, as threads trabalhadoras. Após a inicialização das threads é feito um join na thread leitora e, em seguida, nas threads trabalhadoras. Após o encerramento de todas as threads as estatísticas de execução das tarefas são calculadas e exibidas na tela.

A thread leitora executa um laço em que ela lê tarefas do arquivo de entrada e coloca essas tarefas na fila de tarefas, até que a fila fique cheia. Após a fila de tarefas encher, a thread leitora espera, por meio de uma variável de condição, que uma thread trabalhadora a sinalize no momento em que a fila pode ser preenchida novamente. Quando o arquivo de entrada não possui mais tarefas, a thread leitora coloca uma tarefa para cada thread na fila, indicando o fim das tarefas, e encerra sua execução. Vale notar que caso essas tarefas para cada thread não caibam todas na fila, a thread leitora espera até que ela possa colocar novamente elementos na fila novamente, para, então, inserir todas as tarefas de fim das tarefas na fila, de forma que o tamanho máximo da fila nunca seja ultrapassado.

Já as threads trabalhadoras também executam um loop em que elas lêem uma tarefa da fila de tarefas, realizam os cálculos dessa tarefa e registram as estatísticas da execução dessa tarefa. Quando a fila de tarefas está vazia a thread trabalhadora espera, por meio de uma variável de condição, que a thread leitora coloque novas tarefas na fila. Quando a fila de tarefas pode ser preenchida a thread trabalhadora sinaliza para a thread leitora que ela pode colocar novas tarefas na fila. Já quando a thread trabalhadora lê uma tarefa que indica o fim das tarefas, ela encerra a sua execução.

2.1. Fila de tarefas

A fila de tarefas foi implementada por meio de uma variável global do tipo std::queue da STL do C++. Essa estrutura de dados foi escolhida por suportar exatamente as operações

desejadas: remover elementos do início da fila e colocar elementos no final da fila.

Além disso, essa estrutura possui um uso fácil e eficiente, já que realiza as operações com custo constante, e para usá-la basta declarar a variável e chamar os métodos push, front e pop da estrutura.

2.2. Coleta de Estatísticas

A coleta de estatísticas ocorre em dois pontos da thread trabalhadora: ao buscar uma nova tarefa e ao finalizar a execução de uma tarefa. Quando uma nova tarefa é buscada a primeira coleta de estatísticas acontece, que é a verificação de se a thread trabalhadora encontrou a fila vazia. Logo após a thread obter a tarefa também é guardado o momento em que a thread começou a executar a tarefa.

Após a execução da tarefa, são coletadas a maior parte das estatísticas. Primeiro é guardado o momento que a thread acabou de executar a tarefa. Com o momento de início e de fim da execução da tarefa, é calculado o tempo gasto para executar a tarefa. Após isso, também é atualizado, em variáveis globais, o número de tarefas executadas até o momento, quantas tarefas cada thread executou, o tempo total gasto executando tarefas, o tempo gasto para executar cada tarefa e o número de vezes que alguma thread trabalhadora encontrou a fila de tarefas vazia.

3. Decisões de Projeto

3.1. Sincronização

A principal decisão de projeto foi a sincronização do acesso às seções críticas do programa. Para controlar tal sincronização foram criados dois mutexes: um para controlar o acesso à fila de tarefas e outro para controlar o acesso às variáveis de coleta de estatísticas.

O acesso à fila de tarefas acontece quando a thread leitora inicia sua execução, sendo que ela libera o mutex apenas ao ser encerrada ou ao entrar em espera para ser acordada. Esse acesso também acontece quando uma thread trabalhadora busca uma tarefa na fila. A thread trabalhadora obtém o mutex antes de acessar a fila e a libera apenas após a thread obter uma tarefa da fila, ou ao encontrar a fila vazia e esperar pela thread leitora acordá-la.

Já o acesso às variáveis de coleta acontece no fim da execução de cada tarefa, onde as informações para o cálculo das estatísticas são salvas. Foi usada sincronização nesse trecho devido à possibilidade de mais de uma thread tentar acessar as variáveis de estatísticas ao mesmo tempo.

A sincronização também foi feita usando variáveis de condição. Foram declaradas duas variáveis de condição: uma para controlar quando a thread leitora é acordada para ler novas tarefas e preencher a fila, e outra para controlar quando uma thread trabalhadora é acordada após encontrar a fila de tarefas vazia. A thread leitora espera pela primeira variável e sinaliza a segunda, enquanto as threads trabalhadoras podem esperar pela segunda variável e sinalizar a primeira.

3.2. Codificação

Para o desenvolvimento do código foi usada uma sintaxe mista de C e C++. No geral foi mantido um estilo próximo do C, entretanto diversos recursos de C++ foram usados, como as estruturas de dados queue e vector da STL, a declaração de variáveis dentro do loop for e nas atribuições e também o uso do tipo bool.

Também foi adotado um padrão de definir as variáveis globais constantes com a macro #define. Ou seja, as variáveis globais que representam parâmetros configuráveis do programa são definidas com o #define.

3.3. Cálculo e Exibição das Estatísticas

O cálculo e exibição das estatísticas foram realizados na função main por questão de simplicidade, já que não vi grandes vantagens em criar uma thread separada para lidar com essa funcionalidade.

4. Testes

Para os testes utilizei os seguintes arquivo de testes:

- t, que é o arquivo fornecido com o material para o trabalho, que contém 64 tarefas e que todas as tarefas consistem numa área de 80x60.

- $t256$, que é um arquivo com as tarefas de t repetidas 4 vezes, logo que contém 256 tarefas que têm uma área de 80x60.

- $black$, que é um arquivo com 64 tarefas de uma região escura do gráfico e com uma área em pixels de 80x60.

- $large$, que é um arquivo com 64 tarefas do arquivo t, mas com uma área em pixels de 320x240.

- $large\_black$, que é um arquivo com as mesmas tarefas do arquivo $black$, mas com uma área em pixels de 320x240.

Como meu computador possui um processador i3-9100f, que possui 4 cores, segue os resultados da execução do programa com uma thread e com quatro threads, para todos os arquivos de testes citados:

Instância	Uma Thread			Quatro Threads			
	Tempo Total do Programa (s)	**Tempo Médio por Tarefa (ms)**	**Desvio Padrão do Tempo por Tarefa (ms)**	**Tempo Total do Programa (s)**	**Tempo Médio por Tarefa (ms)**	**Desvio Padrão do Tempo por Tarefa (ms)**	**Desvio Padrão das Tarefas por Trabalhador**
t	9,749	152,328	297,512	2,725	155,234	303,894	5,050
t256	38,922	152,031	297,693	10,198	155,016	303,233	11,247

black	61,250	957,031	24,458	15,493	966,687	6,207	0,000
large	154,669	2416,687	4729,734	42,745	2439,391	4776,219	5,050
large-black	968,336	15130,250	243,433	245,896	15367,203	58,759	0,000

Primeiramente, temos que o speed up para os quatro threads foi, em média, um valor próximo a $3,5$ vezes. Usando a fórmula da Lei de Amdahl podemos aproximar a proporção paralelizável do programa como $\frac{20}{21}$, o que é um valor razoável já que a maior carga do programa acontece no cálculo das tarefas, que é um trecho do programa executado paralelamente.

Outro dado que podemos notar na tabela é o tempo médio gasto por tarefa, que é ligeiramente menor para uma thread. Isso pode ser explicado pelo overhead de escalonamento das threads, o que aumenta um pouco o tempo de execução de cada tarefa.

Também podemos ver que o desvio padrão do tempo gasto por tarefas é similar para cada arquivo de teste, com uma ou com quatro threads. Isso pode ser explicado pelas diferentes características das tarefas de cada conjunto de testes, o que leva a desvios padrão diferentes entre os conjuntos, mas similares mesmo com um número diferente de threads. Mas os desvios não são iguais para um número diferente de threads devido às flutuações do tempo de execução de cada tarefa, que pode acontecer por diversos motivos, desde ruídos ao tempo gasto com escalonamento.

Por fim, temos que a média de tarefas por trabalhador é sempre o número de tarefas do arquivo dividido pelo número de threads. Já o desvio padrão foi 0 para arquivos em que todas as tarefas são idênticas, o que faz sentido. Para os demais temos um pequeno desvio padrão, que indica que algumas threads executaram um pouco mais de tarefas do que outras, devido às tarefas terem complexidades diferentes.

Outra informação, além da tabela, é que para os arquivos de teste $black$, $large$ e $large\_black$, ao executar com quatro threads, a fila de tarefas é encontrada vazia uma vez em algumas execuções. Isso pode ser explicado por essas tarefas serem longas e pelo número de threads não ser muito grande, portanto pode acontecer da thread leitora ser sinalizada para preencher a fila, mas todas as threads executarem uma tarefa antes disso ser feito de fato.