Primeiro exercício de programação: Mandelbrot paralelizado

Júlia Fonseca de Sena - 2018054508

Decisões de projeto

Fila circular

A implementação da fila circular foi simples: um *array* global com tamanho 4 vezes o número de *threads*, uma variável global que indica o índice do começo da fila no *array* e outra que indica quantos elementos há atualmente na lista. Somente duas funções a alteram, a que insere uma tarefa, e a que remove a primeira da fila. Há também uma função que imprime a fila, que foi utilizada somente para *debug* do código.

A função de inserção, que só insere se a fila não estiver cheia (caso contrário retorna -1), coloca a tarefa na última posição, que pode ser encontrada pela soma do índice do começa da fila com o número de elementos atualmente na fila módulo o tamanho máximo da fila, e adiciona 1 à contagem de elementos. Já a remoção, a qual também retorna -1 caso a fila esteja vazia, somente adiciona 1 ao começo e subtrai um do número de elementos na fila. Isso funciona, pois se você aumenta o índice do começo em 1 e retira 1 do tamanho atual da fila, a fila começa 1 índice depois (na tarefa seguinte da fila) e acaba no mesmo índice, ou seja, é a mesma fila sem o primeiro elemento.

Thread mestre (entrada)

A *thread* de entrada lê as tarefas do arquivo recebido como parâmetro e as coloca na fila circular para que as trabalhadoras tenham acesso. Primeiramente, ela popula toda a fila, depois popula os espaços disponíveis quando detecta que o número de tarefas na fila é menor ou igual ao número de *threads* trabalhadoras. Isso é feito por uma função responsável por popular a fila que retorna 1 quando o arquivo acaba. Assim, a variável **eof** recebe o inteiro retornado, e quando for 1, a *thread* passa a colocar os **EOW** na fila.

O indicativo de que uma tarefa é **EOW** é o left ser -1, já que não está dentro do domínio. A *thread* tenta colocar **EOWs** na fila enquanto nem todos forem adicionados, e, se a fila estiver cheia, tenta inserí-los até que o número de **EOWs** restantes não enfileirados seja 0. Após isso, a *thread* finaliza. Assim como nas outras *threads*, toda modificação na fila é guardada por um *mutex*.

Threads trabalhadores (cálculos)

Cada uma das *threads* trabalhadoras possui um id único, o qual é atribuído dentro da própria por um contador guardado pelo *mutex* específico. Esse id é utilizado para salvar valores de cada *thread* no *array* de estatísticas do número de tarefas por *thread*, para que depois seja possível calcular o desvio padrão.

A *thread* pega a primeira tarefa da fila, a remove, e verifica se é **EOW**. Se não for, calcula por meio da função fractal fornecida pelo professor, se for sai do loop e libera a *thread*. Como foi dito anteriormente, as alterações na fila de tarefas são protegidas pelo *mutex*, de forma que não haja problemas de sincronização.

Estatísticas

As estatísticas são colhidas nas *threads* trabalhadoras. Os números de tarefas de cada *thread* são guardados em um *array* na posição do índice que lhe foi designado, para que seja calculado posteriormente o desvio padrão de tarefas por *thread*. Quando se chama a função de remover tarefa, se ela retornar -1, a fila está vazia, então incrementa-se a contagem de filas encontradas vazias.

Coleta-se o total de tarefas na função de popular a fila circular, somando 1 no contador toda vez que se insere uma tarefa na fila. Isso poderia ser obtido também pela soma do número de tarefas de cada *thread*, mas para simplicidade e para checagem de corretude adotou-se também essa variável de total.

Por fim, para medir o tempo, inicia-se logo antes de trancar o *mutex* (para contabilizar o tempo de espera na fila), e finaliza-se após o cálculo do fractal (ou no caso de **EOW**, após ser detectado). A coleta das estatísticas de tempo foi mais complexa, já que não se sabe a média e o total de tarefas no final do trabalho, dificultando o cálculo do desvio padrão, então assumiu-se 2048 como um máximo de tarefas, e criou-se um vetor de inteiros desse tamanho que é preenchido à medida que se realiza as tarefas. Como o vetor em **C** não pode ser aumentado como em **Python** com os *appends*, foi a melhor solução encontrada. Os dados foram obtidos com *clock_gettime* e convertidos de para milissegundos.

As estatísticas só são contabilizadas se de fato for uma tarefa, e não o **EOW**. As equações para média e desvio padrão utilizadas para o cálculo foram as seguintes:

$$m = \Sigma x_i / N \quad d = \sqrt{\Sigma (x_i - m)^2 / N}$$

Onde m é a média, d é o desvio padrão, N é o número de *threads* ou de tarefas, e x é o número de tarefa das *threads* ou tempo de execução das tarefas, dependendo da estatística sendo coletada.

Análise temporal

A máquina onde a análise temporal foi realizada possui um **Intel(R) Core(TM) i7-8565U**, o qual possui 4 núcleos. Assim, foi feita a comparação do tempo gasto com 1 *thread* e com 4. Os valores a seguir foram obtidos com a média das estatísticas obtidas em 10 execuções.

Número de threads	Média do tempo de execução por tarefa	Desvio padrão do tempo de execução por tarefa	Média de tarefa por thread x Média do tempo por tarefa
1	163,547	327,837	10.467,008
4	160,995	314,752	2.575,92
Razão (N=1 / N=4)	**1,016**	**1,042**	**4,063**

Apesar das estatísticas por tarefa serem muito parecidas para ambos números de *threads* (razão bem próxima de 1), multiplicando-se pelo número médio de tarefas por *thread* (já que executam simultaneamente) pode-se obter uma estimativa do tempo total. Percebe-se que o aumento das *threads* para 4 diminuiu basicamente também em 4 vezes.

Então, apesar de não ter muito efeito no tempo de execução por tarefa, percebe-se que o uso das *threads* otimiza o tempo total de execução do programa, visto que as tarefas podem ser feitas de forma simultânea.