perseguição por equipes

Nathan Benedetto Proença
8941276
Victor Sena Molero
8941317

organização do código

A simulação trabalha com duas abstrações principais:

1. pista,
2. ciclista.

Também há uma terceira abstração para gerenciar as voltas

organização da simulação

A simulação em si se organiza em turnos, cada um deles representando um intervalo de 60ms.

Em cada turno, as threads que representam os ciclistas atualizam as informações na estrutura de dados que representa a pista.

pista

A `pista` é o recurso sendo disputado pelo problema simulado.

Trata-se de uma estrutura de dados que contém as informações utilizadas pela simulação.

Há um `mutex` para cada posição possível para os ciclistas.

Se houvesse um `mutex` único para a pista como um todo, a simulação seria essencialmente sequencial.

mudança na representação

Contrário ao que foi especificado no enunciado, optamos por representar cada meio metro de pista como uma posição no vetor, e não cada metro.

Esta decisão foi tomada para melhor representar ciclistas se movendo a 30km/h.

informações armazenadas

A `pista` abstrai a posição dos ciclistas durante a corrida.

Para tal, em cada posição armazena:

Até dois structs do tipo `ciclista_obj_struct` com as informações associadas aos ciclistas.

Dois inteiros, `quantidade` e `atualizados`, que representam, respectivamente, quantos ciclistas existem nessa posição, e quantos deles já foram processados no turno atual.

Ainda há o `mutex` associado àquela posição, e a variável de condição que representa para se aquela posição já foi processada nesse turno.

encapsulamento do acesso

O acesso à `pista` é encapsulado através de funções auxiliares, tanto responsáveis por gerenciar a exclusão mútua, quanto por de fato modificar a informação na estrutura.

`pista_insere` e `pista_remove` são responsáveis por colocar e remover um ciclista, enquanto `pista_livre` testa se é possível avançar para uma posição.

A função `pista_lock` e `pista_unlock` são apenas wrappers para permitir acesso ao mutex e às variáveis de condição de uma maneira mais conveniente.

barreira de sincronização

Há no comportamento da pista uma barreira de sincronização sendo imposta.

Para que um ciclista se mova para uma determinada posição, é necessário que todos os ciclistas que possam impedir ele de fazer isto já tenham sido processados.

A barreira garante que os contadores quantidade e atualizados sejam iguais, fazendo com que a informação consultada pelo ciclista que tenta avançar esteja atualizada.

ciclistas

O `ciclista` é a thread que precisa acessar o recurso compartilhado.

Há diversas informações associadas a eles, tanto descrevendo cada ciclista em específico, quanto o status da corrida como um todo.

a corrida maluca

Armazenamos em `ciclista_n` a quantidade de ciclistas em cada equipe.

A variável `ciclista_tipo` nos diz se estamos na simulação com velocidade uniforme ou na com velocidade variada.

A variável `ciclista_acabou` diz se a corrida terminou.

Também há a função `ciclista_sorteia_quebra` para garantir o comportamento de quebra.

lance armstrong

Para cada ciclista em particular é necessário armazenar as seguintes informações:

Uma `id` para diferenciar cada um.

O índice do `time` o qual ele pertence.

Sua `volta`, `posicao`, `velocidade` e se é possível ultrapassá-lo (`ultrapassavel`).

Se ele está na pista: `ini`,
Se completou o percurso: `fim`,
Ou se quebrou: `quebrado`.

A `thread` responsável por ele.

outra barreira de sincronização

Há também outra barreira de sincronização , mas com relação aos `ciclistas` e a atualização de seus turnos de 60ms.

Esta barreira é implementada através da sincronização por flags, que centraliza a coordenação na thread principal.

Escolhemos esta abordagem pela simplicidade.

quebra dos ciclistas

A função `ciclista_sorteia_quebra` implementa o comportamento descrito no enunciado ao pé da letra.

Caso seja decidido quebrar um ciclista, se escolhe o time e qual deles, tudo de forma aleatória.

avant garde

O avanço de cada `ciclista` é feito pela função `ciclista_avanca`.

Ela apenas olha para a posição em frente ao ciclista e o avança se possível.

As checagens sobre possibilidade de ultrapassagem e espaço livre são feitas pelas funções da pista.

A validade da informação acessada fica garantida pelo `pista_lock`.

a `ciclista_runner`

Ao contrário das demais partes do código aqui apresentadas, a função `ciclista_runner`, que é a executada pela thread, implementa diversos comportamentos, e iremos explicá-la aos poucos.

turnos

O `round` de cada ciclista não representa o turno de 60ms.

Cada turno é implementado em dois `rounds`.

Os pares fazem a execução, e os ímpares permitem a limpeza das flags da pista pelas threads.

fim da corrida

Existem diversas condições que fazem com que um ciclista tenha que parar de correr e precisam ser checadas pelas threads.

1. A quebra do ciclista.
2. A finalização do percurso.
3. A vitória de algum time.

Em todo os casos, o ciclista é removido da pista, e a thread é terminada, além disso, as variáveis fim e ciclista_fim são atualizadas para indicar que esta thread não está mais ativa.

interação com thread principal

As threads dos cicilistas se comunicam com a thread main por meio de flags.

Há uma barreira de sincronização formada pelas variáveis condicionais ciclista_cond_ciclista e ciclista_cond_principal, pela variável global ciclista_round e pela variável round referente a cada um dos ciclistas.

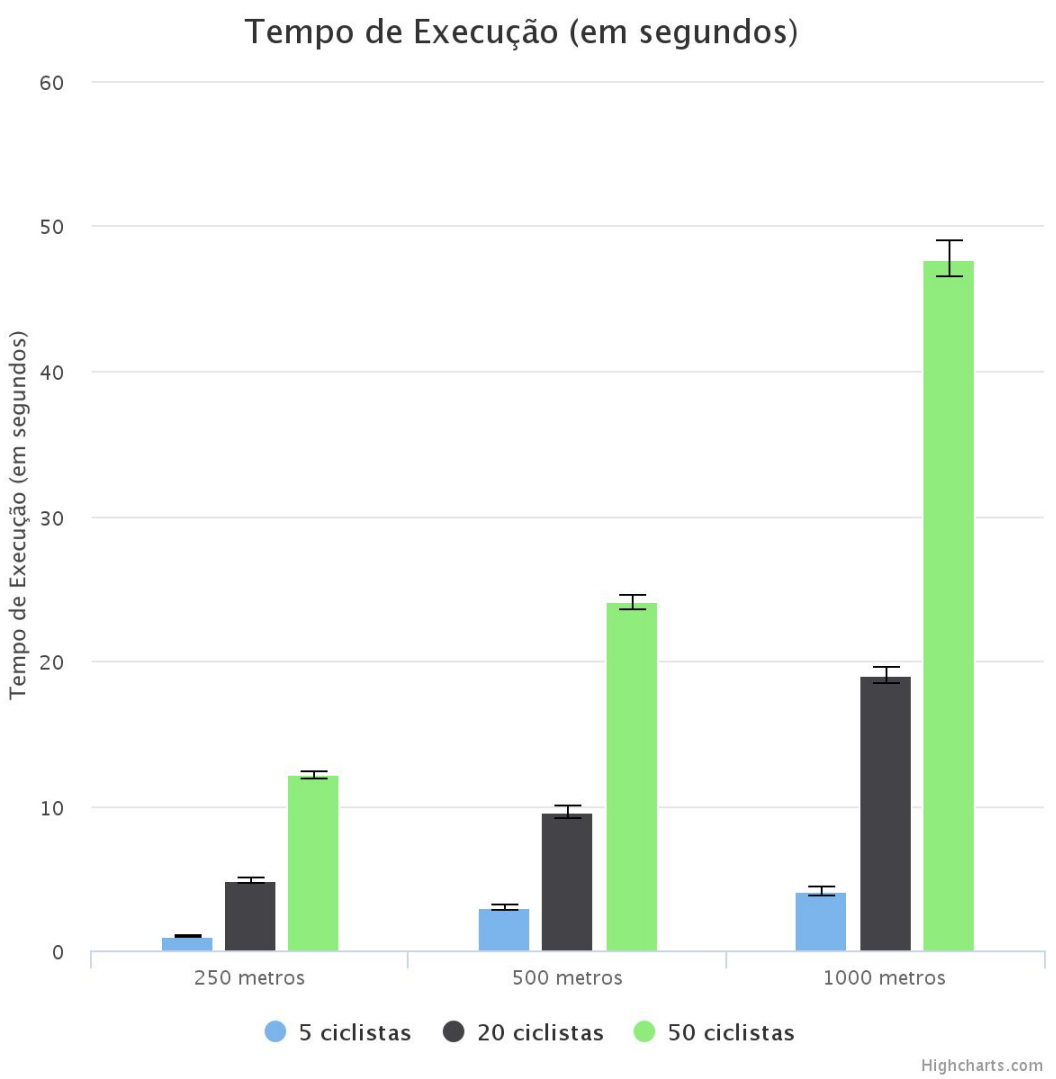
Antes de processar cada rodada, tanto o ciclista quanto a thread principal esperam que todos os ciclistas e a thread principal tenham terminado o round anterior. Após terminar cada um atualiza suas respectivas variáveis round e sinaliza as variáveis condicionais necessárias.

volta

A abstração volta facilita a identificação do fim de uma volta para que seja sorteada a quebra de um ciclista a cada 4 voltas, como especificado no enunciado.

tempo de execução (médias)

	Ciclistas		
	5	20	50
250m	1.01s	4.82s	12.11s
500m	3.02s	9.51s	24.06s
750m	4.05s	18.98s	47.73s



uso de memória (médias)

	Ciclistas		
	5	20	50
250m	1,803Mb	2,182Mb	3,073Mb
500m	1,758Mb	2,234Mb	3,392Mb
750m	1,839Mb	2,488Mb	2,724Mb