EP2

Renê Cardozo - 9797315
Verônica Stocco - 6828626
rene.cardozo@usp.br
veronica.stocco@usp.br

Instituto de Matemática e Estatística
Universidade de São Paulo

Corrida por eliminação

O simulador é composto por dois módulos: aleatorio e pista.

- **aleatorio** contém as funções responsáveis por determinar a velocidade e a probabilidade de quebra dos ciclistas.
- **pista** contém o simulador em si, e será discutido em detalhe nos slides a seguir.

Barreiras, turnos e pistas

O simulador utiliza duas barreiras para gerenciar os turnos da simulação. Desta forma, enquanto as n threads ciclistas restantes aguardam em uma barreira, a barreira referente ao turno seguinte é criada. A operaçao das threads em **void ciclista()** é feita considerando esses dois possíveis turnos (par e ímpar).
Também foi utilizada uma barreira adicional para definir a largada das threads.
A pista é composta por 10 faixas (é possivel editar esse valor em pista.h). Cada linha tem **d** metros, e um **mutex_pos** referente a cada metro / linha da faixa. No início da simulação, os ciclistas são distribuídos nas faixas 0, 2, 4, 6 , 8.

Ultrapassagem

A cada rodada, a posição atual do ciclista é atualizada com **atualiza_posicao**. Para evitar conflitos, são usados 4 mutex, referentes à:

- linha na qual o ciclista se encontra no momento;
- linha para a qual o ciclista pode se mudar caso ocorra uma ultrapassagem;
- próxima faixa (posição) que o ciclista pode ocupar na linha atual;
- próxima faixa (posição) que o ciclista pode ocupar caso ocorra uma ultrapassagem;
  Os mutex referentes às linhas são utilizados para atualizar o número de ciclistas para que este não passe de 5 em cada linha.
  Os mutex de posição são utilizados para travar os metros da pista individualmente.

A eliminação de ciclistas é feita da seguinte forma:

O último ciclista a completar uma volta de número par recebe uma flag em seu ID no vetor **elimina_id[]**. Essa marcação occore no main. A eliminação é executada pela thread referente a esse ID, quando ela for executada.
Quando um ciclista é eliminado, trancam-se os mutex ferentes à linha e faixa que ele ocupava para marcar essa posição como estando livre. Sua colocação é adicionada ao ranking, e outro mutex é utilizado para garantir que apenas uma thread altere o ranking a cada vez.
O critério de eliminação por quebra foi implementado, porém, por apresentar comportamente irregular ele não foi utilizado na implementação final.

Ranqueamento

Optou-se por implementar 2 rankings distintos.

- **Ranking a cada rodada:** registra, no ID de cada ciclista, a sua colocação atual. Paralelo a esse ranking, um vetor de ints atômicos **pos_volta** é utilizado para armazenar a posição do último corredor de cada volta. Quando valor da posição em que se encontra a volta atinge o valor total de ciclistas esperados para aquela volta, o último ciclista é eliminado. Após um ciclista ser eliminado, sua última posição é escrita no ranking de todas as voltas seguintes.
- **Ranking final:** conforme cada ciclista é eliminado, armazena-se sua colocação final na corrida, assim como tempo total, última volta e se sua bicicleta quebrou ou não.

As atualizações nos valores dos rankings sempre são feitas utilizando mutexes referentes a eles.

Bugs conhecidos

O simulador apresenta alguns bugs que não conseguimos corrigir a tempo da entrega.

- **n < 20**: Com um número de ciclistas pequeno o simulador executa corretamente, apenas as duas últimas posições do ranking final não aparecem como o esperado.

- **n > 20**: o simulador apresenta problemas nas últimas voltas da corrida, em especial caso alguma thread alcança as voltas finais muito antes dos outros. Quando isso acontece não ocorrem mais eliminações e os ciclistas remanescentes ficam ativos no loop até que o simulador seja interrompido manualmente. Consequentemente os rankings não são exibidos da maneira correta.

Este último bug pode ser visto melhor caso seja ativada a segunda opção de debug <debug-ranking> e desativada a primeira que utiliza a função **print_pista** que pode ser feita comentando o trecho da main responsável pelo primeiro debug.

Máquina Utilizada

Para realizar os testes foi utilizada uma máquina com um processador de 6 núcleos Intel I7-9750H e 64GB de RAM. O sistema operacional utilizado foi o Pop!OS 20.10, baseado em Ubuntu.

Gráficos: 10, 30, 100 ciclistas numa pista de 1000 m

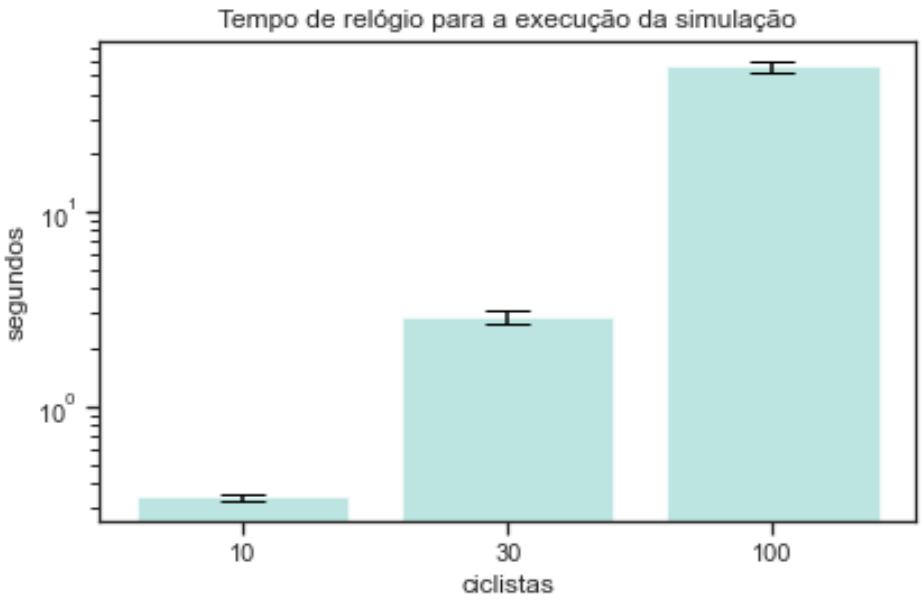

Figure: Tempo de execução

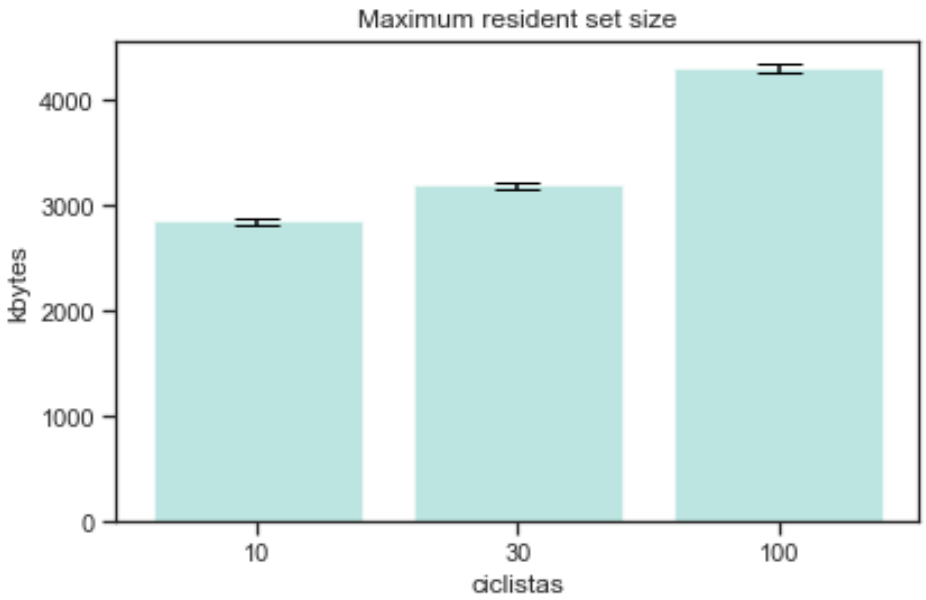

Figure: Uso de memória

Como esperado, aumentar o número de ciclistas / pthreads em execução levou a um aumento considerável no tempo de execução da simulação. Contudo, não era esperado que este aumento ocorresse de forma exponencial. Também era de se esperar que a criação de threads adicionais não aumentasse tanto o uso de memória, a qual é dominada, em especial, pela alocação da pista e do ranking de ciclistas.

Gráficos: 500, 5000, 50000 metros de pista com 20 ciclistas

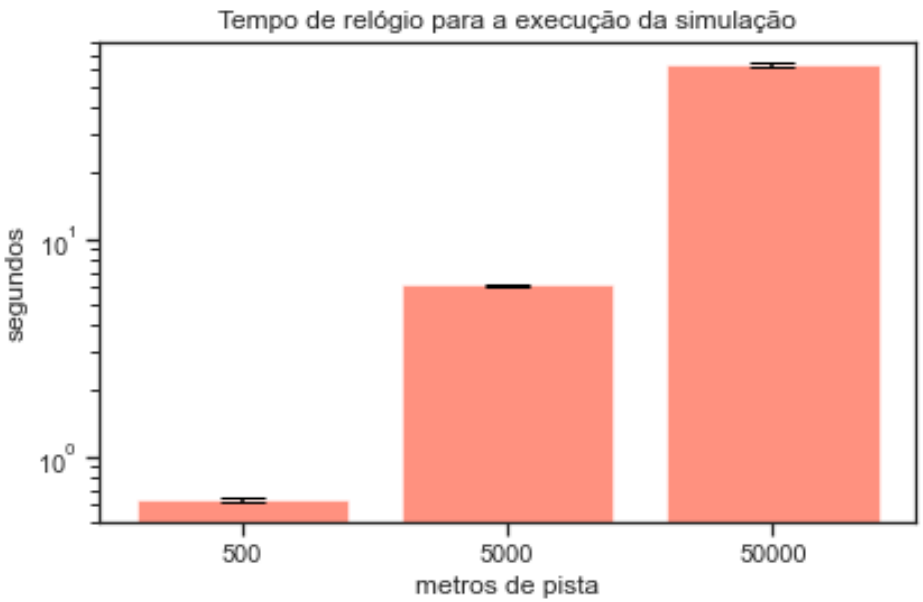

Figure: Tempo de execução

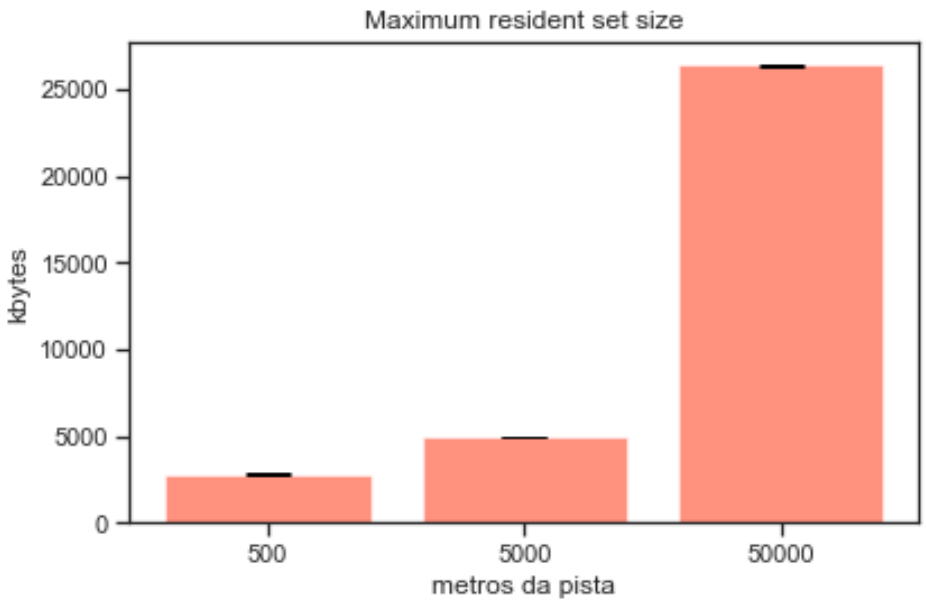

Figure: Uso de memória

Mais uma vez, os resultados foram de acordo com o esperado. O aumento no tamanho da pista levou a um crescimento considerável do tempo de execução da simulação, e a um aumento exponencial no uso de memória.

Gráficos: mudanças de contexto

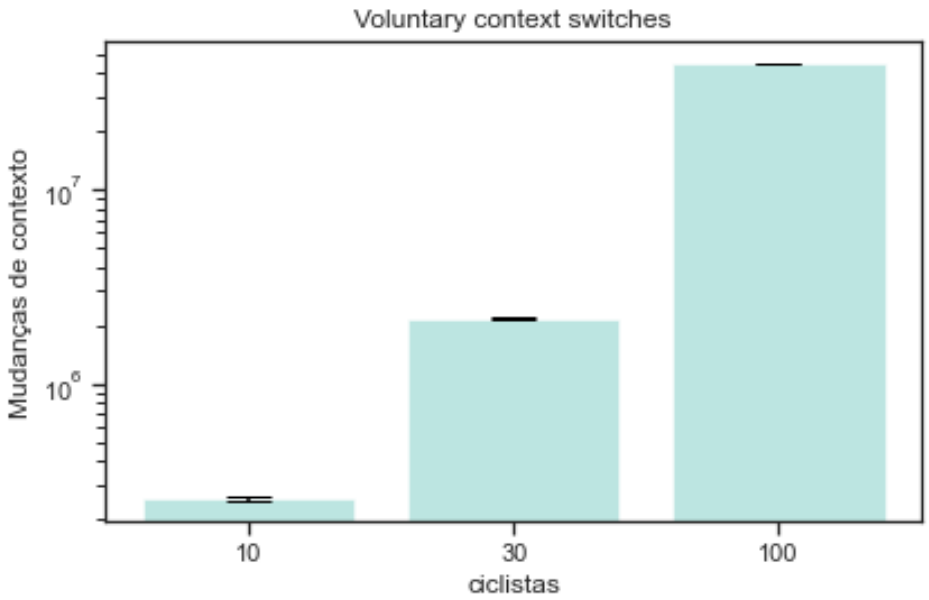

Figure: Alterando número de ciclistas, com d = 1000

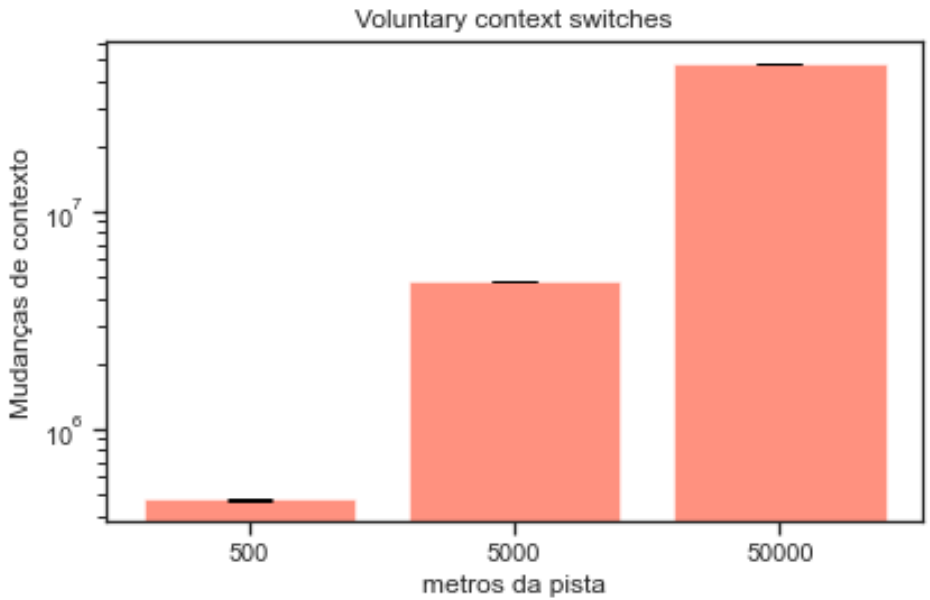

Figure: Alterando o tamanho da pista, com n = 20

No caso do primeiro gráfico, quanto mais threads ativas, mais mudanças de contexto devem ocorrer para que todas completem sua execução. No caso do segundo, o número de threads era fixo, porém o número de iterações que elas precisavam completar para finalizar suas execuções era muito maior.
O aumento na quantidade de mudanças de contexto ajuda a explicar a exponencialidade do tempo de execução.