

## EP2

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

Instituto de Matemática e Estatística  
Universidade de São Paulo

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 operação 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 linhas (é possível editar esse valor em `pista.h`). Cada linha tem **d** metros, e um **mutex\_pos** referente a cada metro / faixa da linha. No início da simulação, os ciclistas são distribuídos nas linhas 0, 2, 4, 6, 8.

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;

# Eliminação de ciclistas

A eliminação de ciclistas se dá de duas formas:

- **Por quebra:** a chance de quebra é definida aleatoriamente, conforme as especificações do enunciado.
- **Por ser o último:** o último ciclista a completar uma volta de número par recebe uma flag em seu ID no vetor **elimina\_id**. Essa marcação ocorre 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 referentes 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 de cada vez.

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.
- **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.

- $n < 30$ : o ranking é impresso, mas a execução do simulador precisa ser interrompida manualmente. Algumas vezes as informações do último ciclista não são devidamente registradas, e o programa imprime que seu tempo e total de volta foram 0.
- $n > 30$ : o simulador funciona como esperado até restarem em torno de 8 ciclistas na corrida. Quando isso acontece, não ocorrem mais eliminações. Os ciclistas remanescentes ficam ativos em loop até que o simulador seja interrompido manualmente.

# Máquinas Utilizadas



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

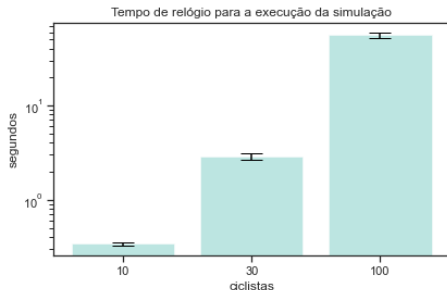


Figure: Tempo de execução

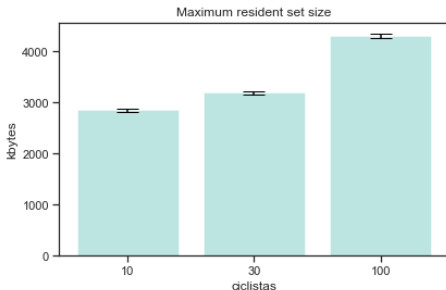


Figure: Uso de memória

Como esperado, aumentar o número de ciclistas / pthreads em execução levou a um aumento considerável no tempo de execução da simulação.

Também era de se esperar que a criação de pthreads adicionais não aumentasse tanto o uso de memória.

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

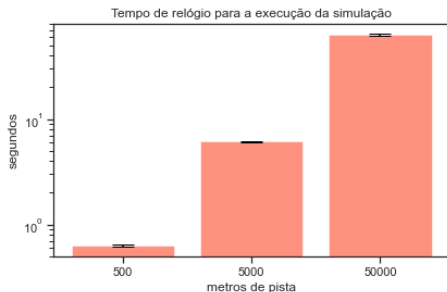


Figure: Tempo de execução

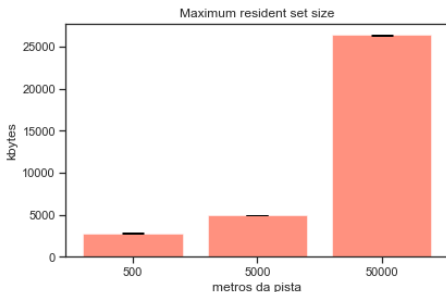
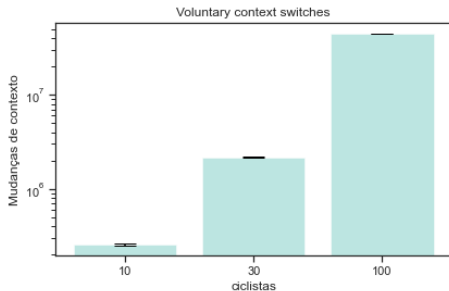


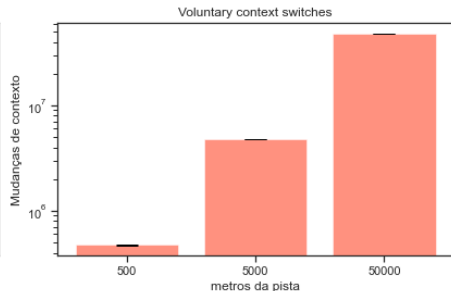
Figure: Uso de memória

Mais uma vez, os resultados foram de acordo com o esperado. O aumento no tamanho da pista levou a um aumento considerável no 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 o aumento na demora para execução das simulações.