

Exercício Programa 2

Corrida por eliminação

Lucas Paiolla Forastiere, 11221911

Marcos Siolin Martins, 11221709

IME-USP

05 de novembro de 2020

Detalhes de Implementação - Os competidores

- Cada ciclista possui uma struct própria com todas as informações necessárias para a corrida e as estatísticas;
- Essas structs ficam armazenadas em um vetor global chamado `ciclistas`;
- Enquanto isso, as threads de cada ciclista ficam em um vetor à parte chamado `threads`.

Detalhes de Implementação - A pista

- A pista é uma matriz de inteiros com tamanho máximo de `D_MAX` (2000) por `FAIXAS` (10). Se `pista[i][j]` é 0, então não há nenhum ciclista no metro i , faixa j , caso contrário, o número dessa posição é o ciclista que se encontra nela;
- Além disso, existe uma matriz `mutex_pista` de mesmas dimensões com um *mutex* para cada posição da pista.

Obs.: Existe uma constante chamada `COM_COR` em `util.c` que, quando alterada para 1, realiza a depuração da pista colorindo os ciclistas na saída.

Detalhes de Implementação - O movimento - 1

- Cada ciclista é responsável por seu próprio movimento, liberando ou travando os *mutexes* necessários. Sempre que precisamos checar uma posição da pista, travamos o *mutex* correspondente, fazemos as verificações e depois o liberamos;
- Se a posição da frente está livre, então ele simplesmente anda para frente;
- Se ela está ocupada por um ciclista que ainda não fez sua ação, então esperamos por ele (liberando o seu *mutex*);

Detalhes de Implementação - O movimento - 2

- Se ele está a mais que 30 Km/h e não conseguiu andar para frente, então ele tenta ultrapassar (verificando as faixas mais externas);
- Na verificação, se ele não conseguir travar o *mutex* por qualquer motivo, então tenta a próxima faixa mais externa.

Detalhes de Implementação - Turnos e Barreiras - 1

- Nós dividimos as interações dos ciclistas em **turnos**. Cada final de turno é protegido por uma barreira, para garantir que todos os ciclistas estão sincronizados em seus turnos;
- Os turnos em que um ciclista se movimenta são definidos de acordo com sua velocidade;
- Ciclistas a 30 Km/h se movimentam de 6 em 6, enquanto os a 60 se movimentam de 3 em 3 e os a 90, de 2 em 2;
- Dessa forma, cada **turno** representa 20 ms de simulação;

Detalhes de Implementação - Turnos e Barreiras - 2

- Além disso, na barreira, temos uma região que é responsável por ações de sincronização, como remover ciclistas eliminados, refazer a barreira (pois estamos utilizando a do `pthread`), incrementar o tempo atual da simulação e imprimir a pista caso necessário;
- Portanto, não foi necessária uma thread coordenadora, já que os próprios ciclistas são responsáveis pela sincronização.

Detalhes de Implementação - Linha de Chegada - 1

- Sempre que um ciclista cruza a linha de chegada, nós verificamos se ele precisa ser eliminado, se quebrou ou se já terminou a corrida;
- Assim que ele cruza, nós adicionamos ele a uma lista ligada que dá as classificações referentes à volta que ele acabou de completar. Assim sabemos volta a volta as classificações;
- Além disso, existe uma outra lista ligada referente a quantos ciclistas quebraram em uma determinada volta. Caso ele quebre, adicionamos ele na lista ligada daquela volta;

Detalhes de Implementação - Linha de Chegada - 2

- Para eliminar um ciclista, nós mantemos uma variável `ult` referente à volta posterior à última completa. Assim sendo, quando `ult` é uma volta par, devemos eliminar o último a completá-la;
- Quando um determinado ciclista está completando sua `ult`-ésima volta, devemos verificar se a volta `ult` terminou (isto é, se todos os ciclistas já a completaram);
- Para saber se ela está completa, verificamos se o número de ciclistas que já completaram `ult` menos o número de ciclistas que quebraram em `ult` é igual ao número de ciclistas que completaram `ult-1`, menos 1 caso `ult-1` seja uma volta par, isto é, alguém foi eliminado;

Detalhes de Implementação - Linha de Chegada - 3

- Quando detectamos que a volta `ult` foi completada, devemos imprimir seu ranking e verificar se alguém deve ser eliminado;
- Se sim, então marcamos o último ciclista a passar por `ult` como eliminado nessa volta e precisamos verificar agora se a volta `ult+1` também já está completa (caso o último a passar pela volta `ult` estiver *muito* atrás na corrida);

Detalhes de Implementação - Linha de Chegada - 4

- Por fim, precisamos mudar as velocidades dos ciclistas. Para as velocidades 30 e 60 Km/h é fácil, basta fazer como pedido no enunciado;
- Já para determinar se alguém estará a 90 Km/h, precisamos prever que determinado ciclista está nas duas últimas voltas;
- Para fazer isso, nós mantemos a variável global `max_voltas`, que é dada pela fórmula $\text{max_voltas} = \text{ult} + 2 * (n_{\text{ult}} - 1)$, onde n_{ult} é a quantidade de ciclistas que passaram para a volta `ult+1`;
- Com ela, nós sabemos exatamente quantas voltas a corrida terá caso todos que estão correndo não quebrem e, assim, conseguimos determinar se alguém está nas duas últimas voltas;

Detalhes de Implementação - Linha de Chegada - 5

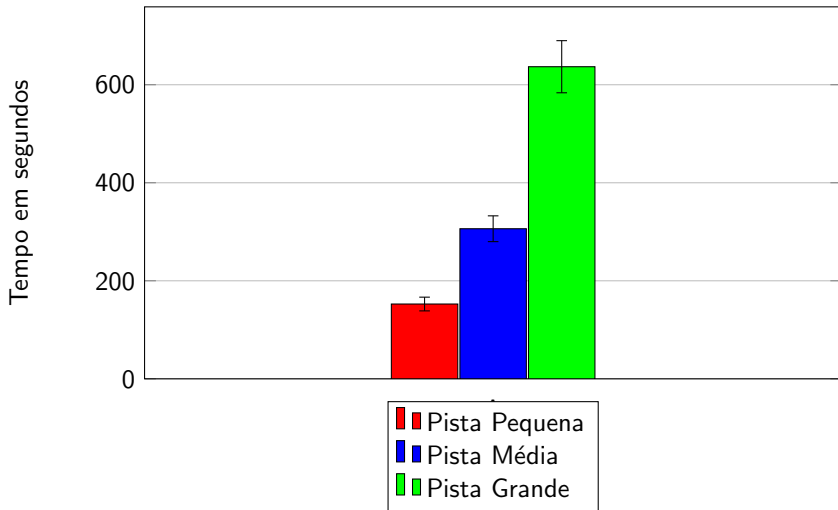
- Para decidir se haverá alguém correndo a 90 Km/h, realizamos um teste no início do programa usando a probabilidade de 10%;
- Caso um ciclista comece a correr a 90 Km/h, mas já deveria ser eliminado em uma volta anterior, então, ao ser eliminado, o próximo que entrar nas duas últimas voltas, correrá a 90 Km/h.

Realização das Medições

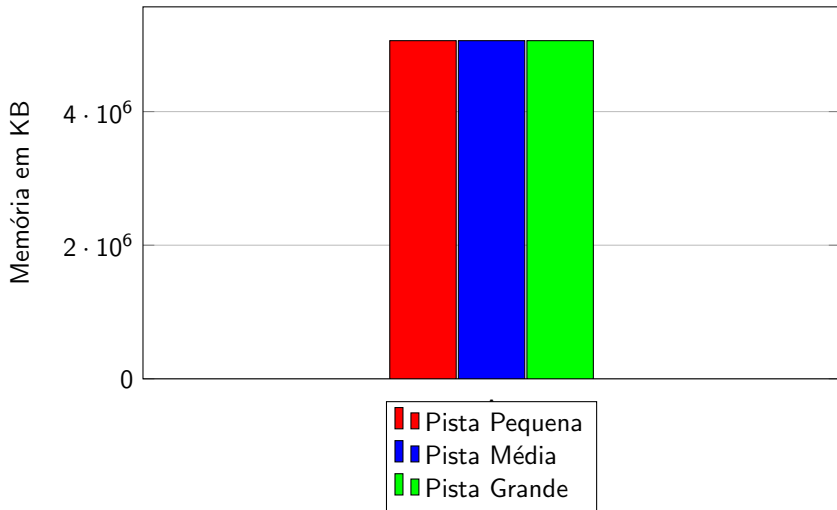
- Para medir tempo consumido, usamos o comando `time` do `bash`;
- Para medir a memória consumida usamos o comando `pmap` passando o *PID* do processo.

Os resultados que obtivemos foram os seguintes:

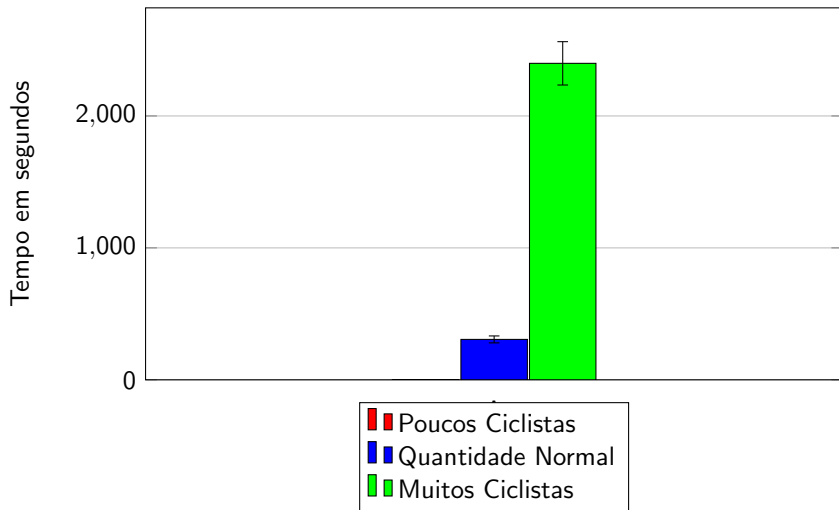
Quantidade Normal de Ciclistas - Tempo



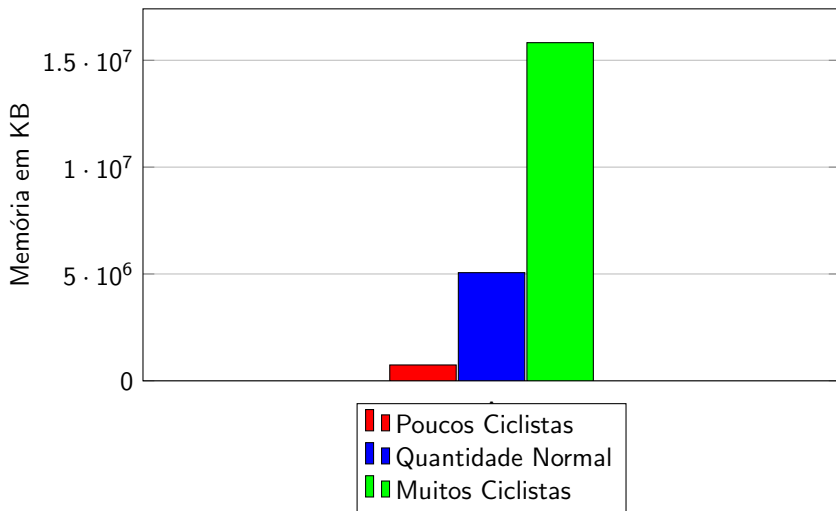
Quantidade Normal de Ciclistas - Memória



Tamanho Normal de Pista - Tempo



Tamanho Normal de Pista - Memória



Conclusões - Memória

- Pelos resultados, observamos que o uso de memória não varia com alterações no tamanho da pista. Isso era esperado, pois a pista é alocada estaticamente e nenhuma memória extra é utilizada quando aumentamos o seu tamanho;
- Já nos aumentos da quantidade de ciclistas, o uso de memória aumentou drasticamente, mas foi um valor igual para todos os 30 testes de cada quantidade. Nós esperávamos que houvessem mudanças, pois alocamos as listas ligadas dinamicamente de acordo com os classificados de cada volta e os ciclistas que quebram (o que não é igual em cada teste).

Conclusões - Tempo

- Observamos aumento do tempo tanto ao aumentar o número de ciclistas como ao aumentar a pista;
- O aumento do número de ciclistas impactou muito mais o tempo de execução do que o da pista.

Obs.: No slide 16, o tempo médio para poucos ciclistas foi de 0.5 segundos com intervalo de confiança de 0.29 segundos. Por conta disso, não foi possível a visualização no gráfico.

Obrigado!

Lucas e Marcos