EP2

MAC0422 - Sistemas Operacionais
Segundo Semestre (2020)

Nome: Pedro Fernandes nUSP: 5948611

Nome: Thiago Jose Benitez Pena nUSP: 6847829

Objetivo

Criar uma simulação de uma modalidade de competição de ciclismo denominada *Miss and out*. Implantando conceitos importantes visto em aula como barreiras para evitar dessincronização tanto como outros recursos para evitar acesso equivocado em seções críticas. O simulador deve atender todos os requisitos do enunciado como também decisões de projeto que levem a uma modelagem coerente da simulação da competição real.

Divisão dos códigos-fontes

- **ep2.c**
  - Código main, lê entrada, cria estrutura da pista e ranks, threads do coordenador e dos ciclistas, inicializa os mutex e as variáveis globais. Também contabiliza informações de tempo e memória se necessário.

- **rank.c** - rank.h
  - Códigos responsáveis pela estrutura de dados informativas da prova simulada pelo programa tanto em um nível de coleta de dados como impressão de outputs.

Divisão dos códigos-fonte - Parte 2

- **thread_ciclista.c** - thread_ciclista.h
- **thread_coordenador.c** - thread_coordenador.h
    - Códigos de funcionamento das threads tanto as dos diversos ciclistas como a coordenador, denominada “juiz” que realiza organização e barreiras.

- **tools.c** - tools.h
    - Código contendo funções auxiliares, como a geração de números aleatórios.

Funcionamento do programa

Após os ciclistas serem posicionados logo atrás da linha de largada é iniciado um ciclo de funcionamento que corresponde em uma ação por thread de ciclista (“competidor”) que é verificada por uma thread coordenadora (“juiz”).

A ação do ciclista pode ser :

- Avanço.
- Meio-avanço.
- Avanço com ultrapassagem.

Funcionamento do programa - Parte 2

Sendo a rodada de ações dos ciclistas completadas, a thread coordenadora fica responsável pela **sincronização**, ou seja, não haver uma thread privilegiada realizando mais de uma ação por turno, tanto quanto os cuidados da **simulação** da prova, sendo eles:

- Verificar números de voltas completadas pelo primeiro colocado e último colocado.
- Usar a volta do último colocado como critério de eliminação (*Miss & out*).
- Eliminar ciclistas que quebraram.
- Eliminar a thread com o uso da função **pthread_cancel**.
- Atualizar informações de rank da prova tanto para efeitos de impressão de output como, também, base de decisão para as eliminações.

Funcionamento do programa **- Parte 3**

Na struct de um ciclista **p**, a sua velocidade pode assumir os seguintes valores:

- **p->velocidade = 1**, se o ciclista está a 30km/h (1m/120ms)
    - Precisa de 2 intervalos de 60ms para percorrer 1 m.
    - Precisa de 6 intervalos de 20ms para percorrer 1 m.
- **p->velocidade = 2**, se o ciclista está a 60km/h (1m/60ms)
    - Precisa de 1 intervalo de 60ms para percorrer 1m.
    - Precisa de 3 intervalos de 20ms para percorrer 1 m.
- **p->velocidade = 3**, se o ciclista está a 90km/h (1m/40ms)
    - Precisa de 2 intervalos de 20ms para percorrer 1 m.

Funcionamento do programa - Parte 4

- Controlamos quantas iterações da barreira de sincronização os ciclistas precisam ficar parados para avançar uma posição da seguinte forma:

  p->dt = dt_base;

- Sendo **dt_base = 6**, se os intervalos de tempo forem de 20ms (se houver algum ciclista a 90km/h), ou **dt_base = 2**, se os intervalos de tempo forem de 60ms. Assim, a cada iteração em que o ciclista anda uma parcial de 1 m, fazemos:

  p->dt = p->dt - p->velocidade;

- E o ciclista só avança uma posição quando **p->dt == 0**

Funcionamento do programa **- Parte 5**

- Na **largada**, os ciclistas são posicionados antes da linha de chegada, partindo da última posição da pista.
- Cada ciclista tem uma contagem própria do número de voltas completadas inicializada com valor -1 para a largada.
- A contagem de voltas globais da corrida é feita seguindo a contagem de voltas do primeiro e último colocados. Utilizamos duas variáveis locais da thread coordenadora: **maxVolta** e **minVolta**, que calculam o valor máximo e o mínimo de voltas dos ciclistas ativos, respectivamente.
- Uma outra variável local da thread coordenadora, **ultimaVoltaDeEliminacao**, é responsável por verificar quando **minVolta** é alterada entre barreiras de sincronização, iniciando então a verificação das condições para eliminação dos ciclistas.

Critérios de modelagem para a simulação

Alguns critérios de projeto foram decididos levando em consideração características da competição real, enunciado e interações de questões trazidas pelos alunos e respondidas pelo professor no fórum da disciplina.

- **Critérios de Avanço**

- **Critérios de Ultrapassagem**

- **Critérios de Sorteio**

Critérios de modelagem para a simulação
- Avanço

A pista foi dividida em “casas” de 1m, sendo uma avanço normal de uma casa feito a cada rodada por um ciclista que esteja a 60 km/h, ou a cada duas, por um ciclista que esteja a 30 km/h sendo registrado uma subtração de uma variável correspondente ao tempo necessário para cobrir o espaço de 1m.

Como a simulação torna-se mais granular nas duas últimas voltas esse avanço de torna-se mais lento, sendo comprido em uma rodada de threads apenas por um eventual ciclista a 90 km/h.

OBS: variável tempo conta iterações.

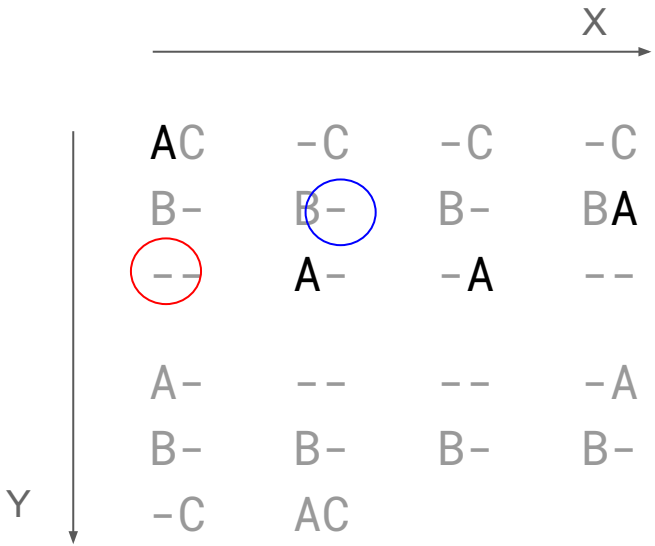
- Se cada iteração vale 60ms (dt_base = 2) -> tempo*60ms
- Se cada iteração vale 20ms (dt_base = 3) -> tempo*20ms (se tiver 90km/h)

Critérios de modelagem para a simulação

- Ultrapassagem

Há três condições para ocorrer ultrapassagem, sendo elas:

- O ciclista a frente já tenha realizado sua ação
- Que haja espaço na coluna y logo abaixo (mais externa) do ciclista que está realizando a tentativa.
- Que haja espaço na coluna y + 1 logo abaixo (mais externa) do ciclista que poderá ser ultrapassado

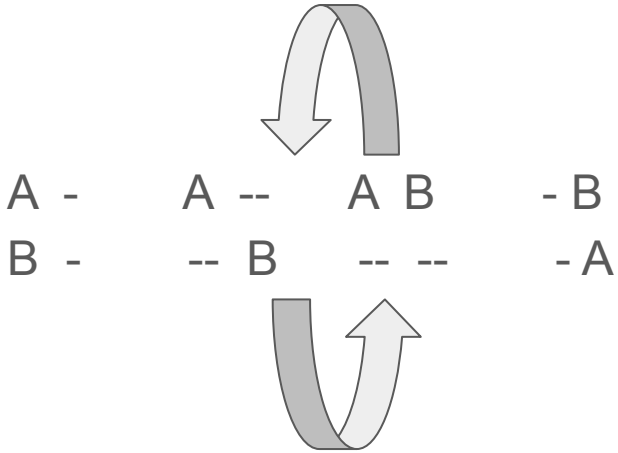


Critérios de modelagem para a simulação - Ultrapassagem **- Parte 2**

Um fenômeno da implementação é um efeito de rotação que ocorre devido ao ciclistas procurarem a pista interna durante as interações

A -
B -

- B
- A



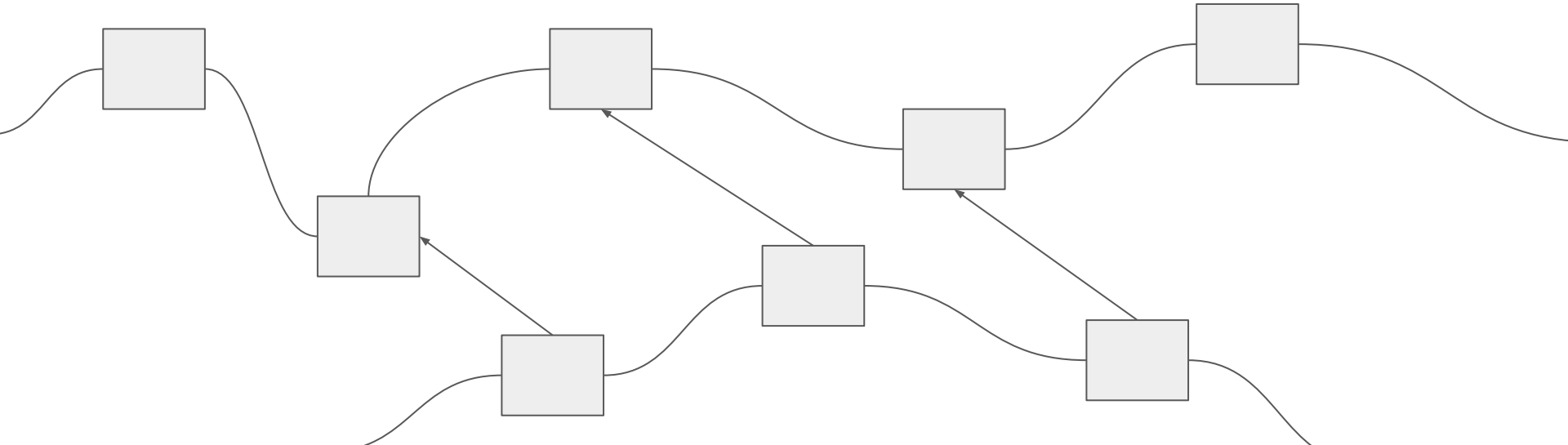
Critérios de modelagem para a simulação
- Sorteio

Além dos sorteios definidos pelo enunciado, com suas devidas probabilidades, deixamos o critério de desempate, tanto de conclusão de volta emparelhada, último colocado emparelhado e volta final, como um efeito de “condição de corrida” benigna, isto é, o S.O. em sua alternância natural fará que uma thread seja executada por último, sendo “rankeada” por último e portanto escolhida.

- OBS: “benigna” pois não ocorre acesso dessincronizado de seção crítica, somente, imprevisibilidade (no caso, desejada).

Estrutura de dados

Levando em consideração que o nosso programa deve ter um desempenho satisfatório independentemente da quantidade de competidores e dimensão de pista foram pensadas estruturas que pudessem responder rapidamente às diferentes demandas



Estrutura de dados 1 - Ciclistas

Foi utilizada uma Lista ligada de struct Ciclista.

- **Motivo:** facilidade de remoção de um ciclista eliminado ou por quebra.

- **Objetivo**: em suma, guarda atributos úteis dos ciclistas, tanto valores numéricos como flags informativas. A vantagem desta implementação é a liberdade de acesso a essas informações tanto pela thread do próprio ciclista como a de outros ciclistas e coordenador.

```
typedef struct Ciclista ciclista;
struct Ciclista {
    int num;
    int px,py;
    int arrive;
    int Continue;
    int voltas;
    int velocidade;
    int dt;
    pthread_t id;
    ciclista *prox;
    bool roundFeito;
    bool quebrou;
};
```

Estrutura de dados 2 - Ranking dos ciclistas

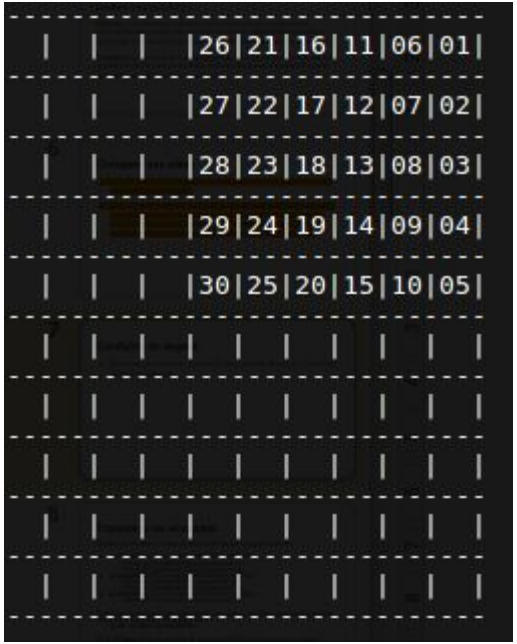
Foi feito uma estrutura (rank) de ranking (para uma dada volta) e uma lista ligada de Rank (ListaRank).

- **Motivo:** A estrutura de lista ligada permite remover dados de voltas passadas quando não forem mais necessárias, evitando assim usar memória em grande quantidade quando houverem muitas voltas.
- **Objetivo**: ListaRank foi usada para poder armazenar para cada volta às informações das posições dos ciclistas e os tempos. Aproveitamos também o struct Rank para armazenar as informações de rank final e das quebras.

Condições de largada

Posicionamos os ciclistas para largada partindo da última posição da pista (1 metro antes da linha de chegada). Os ciclistas são posicionados em grupos de 5 nas posições do vetor pista de índices d-1, d-2, d-3 e assim por diante.

A imagem ao lado mostra um exemplo para d = 10 e n = 30.



Determinação da últimas 2 voltas

Objetivo: saber com antecedência quais são as últimas 2 voltas é importante para (1) sortear um ciclista para pedalar a 90km/h e; (2) interromper a thread do vencedor no momento em que ele terminar a última volta,

Inicializamos o **nVoltasTotal** (número de voltas totais da prova) como:

nVoltasTotal = 2*n - 2

Para cada quebra que ocorrer, **nVoltasTotal** é decrementado em 2. Logo, em qualquer instante, temos:

nVoltasTotal = 2*n - 2 - 2*nQuebras

Determinação da últimas 2 voltas - Parte 2

Quando **nCiclistasAtivos <= 5** , não ocorrerão mais quebras, então **nQuebras** já terá atingido seu valor definitivo e podemos determinar a última volta com precisão.

OBS: Caso haja um primeiro colocado muito rápido e ele esteja muitas voltas à frente de muitos ciclistas, pode ocorrer de ele já ter passado de uma volta que não era para ser a última. Nessa situação, se ocorrerem muitas quebras, sua volta atual poderá ser posterior a nova última volta calculada. Nesse caso ele será considerado vencedor no instante em que soubermos que ele já terminou a nova última volta.

Condições para um ciclista pedalar a 90 km/h

Se o 1º colocado inicia as duas últimas voltas, sua volta local será
p->voltas == nVoltasTotal - 2

Vamos chamar o 1º colocado nesse instante de **ciclistaA** e o segundo colocado de **ciclistaB**.
Vale destacar que ainda não temos certeza de quem é o ciclistaB.

Nesse momento, se for decidido que haverá um ciclista a 90km/, a simulação passa a ocorrer em intervalos de tempo de 20ms e fazemos o sorteio:

- Se o **ciclistaA** for o sorteado, sua velocidade muda para 90km/h nesse instante;
- Se o **ciclistaA** não for o sorteado, esperamos até que um 2º ciclista (**ciclistaB**) alcance as duas últimas voltas para alterarmos sua velocidade para 90km/h.

Método de experimento

- Foi escolhido como parâmetros de testes as distâncias de pista:
  - 250m (pequena)
  - 500m (média)
  - 1000m (grande)
- e número de ciclistas competidores:
  - 10 (pequeno)
  - 100 (média)
  - 300 (grande)

Método de experimento - Parte 2

- Foram realizadas 30 medições, em cada máquina, para cada relação d x n, ou seja, cada parâmetro de distância com um do parâmetro de competidores. Ou seja, no total, cada máquina realizou 30*9 = 270 medições

- Todas encaminhadas consecutivamente via script de bash sendo feitas em máquinas com temperaturas estáveis, todavia competindo com outros processos reais dos respectivos S.O.s

Método de experimento - Parte 3

Para as medições usamos uma mistura de funções, sendo elas

- **getrusage(2)** (https://man7.org/linux/man-pages/man2/getrusage.2.html)
  - Gera uma struct com diversos valores úteis, mas principalmente o uso máximo de memória (física e virtual) que, quando usado com a flag **RUSAGE_SELF** retorna o valor requisitado fa thread pai e seus filhos.

- **clock_gettime(3)** (https://man7.org/linux/man-pages/man3/clock_gettime.3.html)
  - Devido a estrutura do getursage não coletar o tempo de execução real, necessitamos de uma função que acessasse o tempo externo do programa, isto é, o tempo de relógio (wall-clock), e é exatamente isso que clock_gettime faz quando implantado com a flag **CLOCK_REALTIME**.

Máquinas do experimento

Máquina [A]

- 1 CPU - 12 processadores lógicos

  Ryzen 1600 - 3.5Ghz (Overclock)
  - 6 núcleos físicos - 12 lógicos

- 16 GB RAM 2900mhz
- SSD
- Ubuntu 16.04.7 LTS

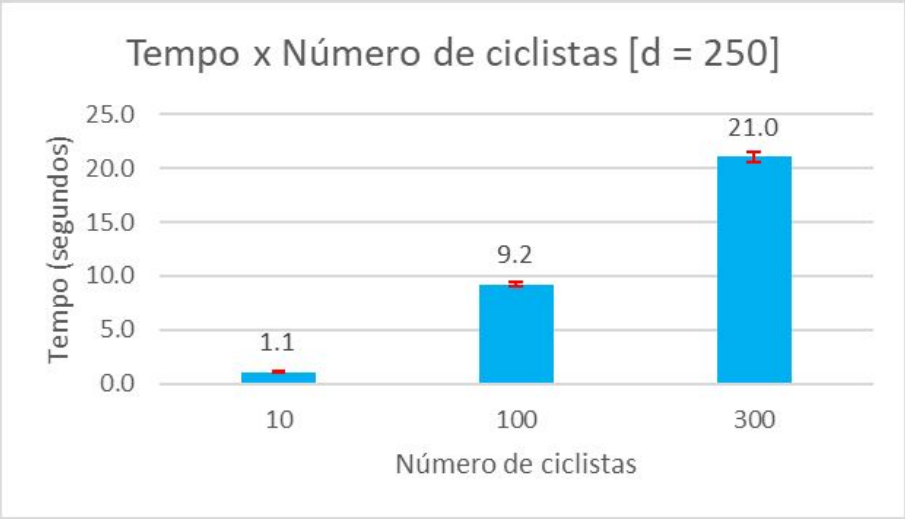
Máquina [B]

- 1 CPU - 8 processadores lógicos

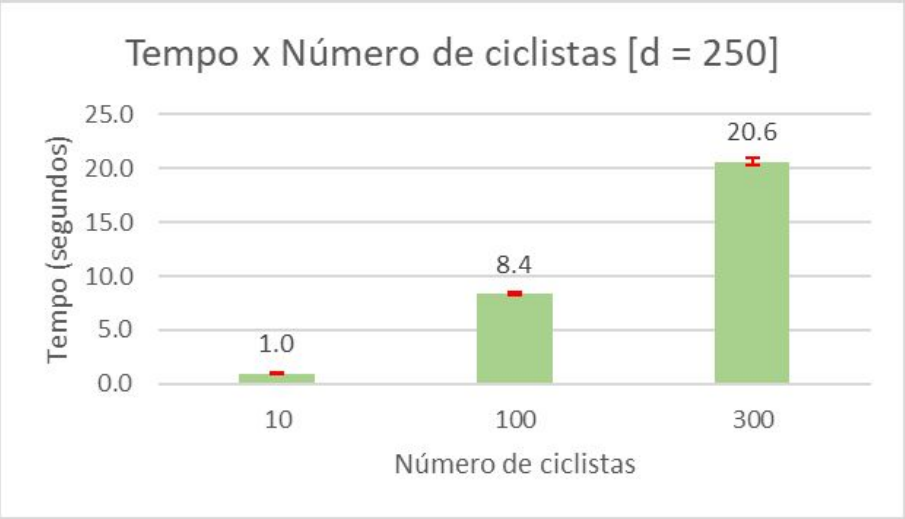
  (Ryzen 3300X - 3.9Ghz
  4 núcleos físicos - 8 lógicos)

- 16 GB RAM 3000mhz
- SSD
- Ubuntu 20.04.1 LTS

Gráficos - Tempo X Número de ciclistas

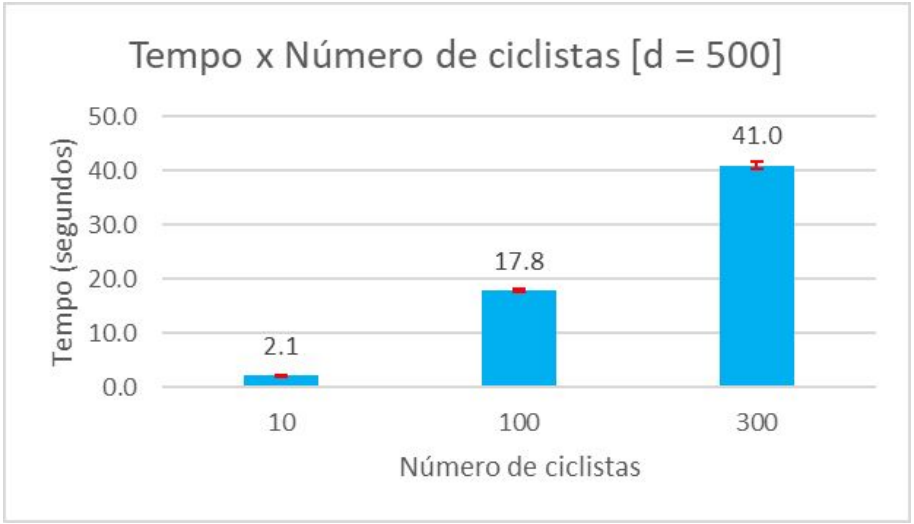


Máquina A

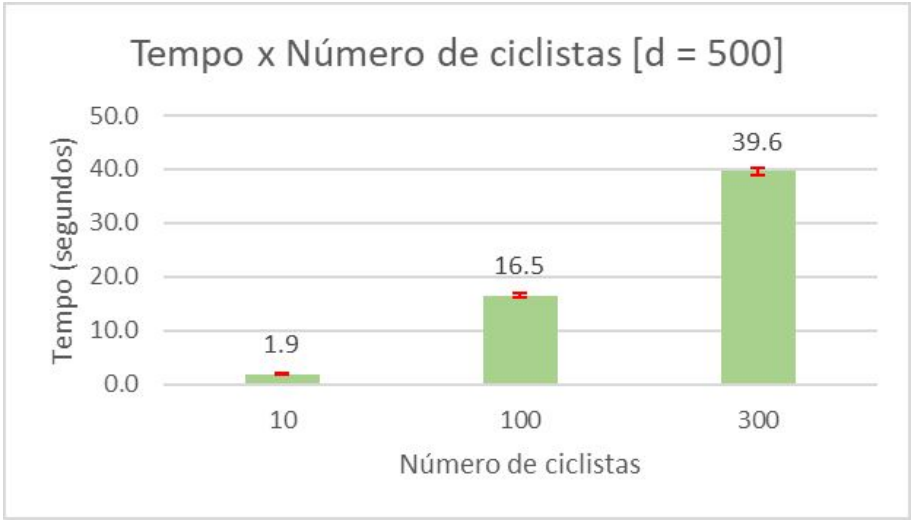


Máquina B

Gráficos - Tempo X Número de ciclistas

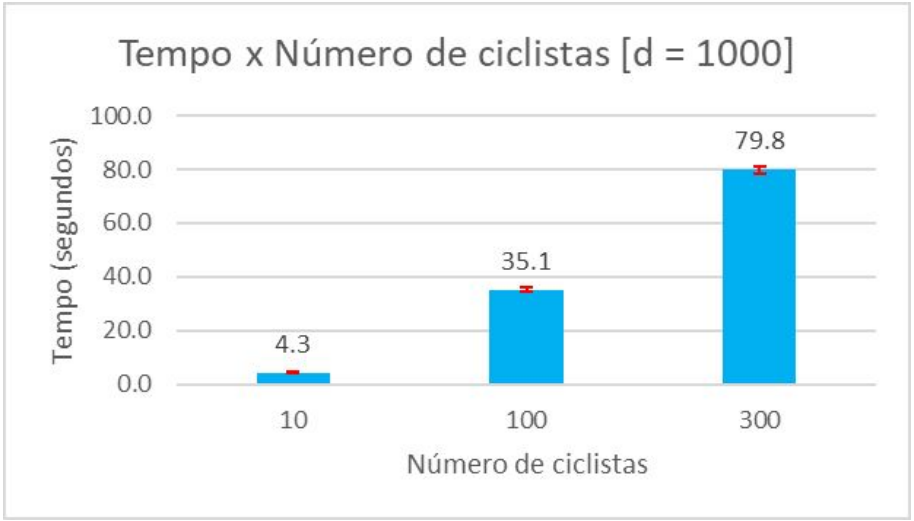


Máquina A

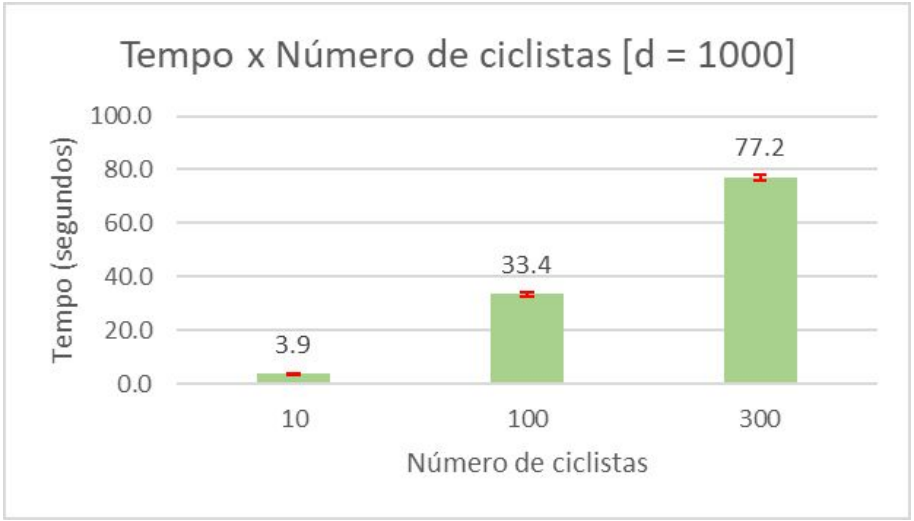


Máquina B

Gráficos - Tempo X Número de ciclistas



Máquina A

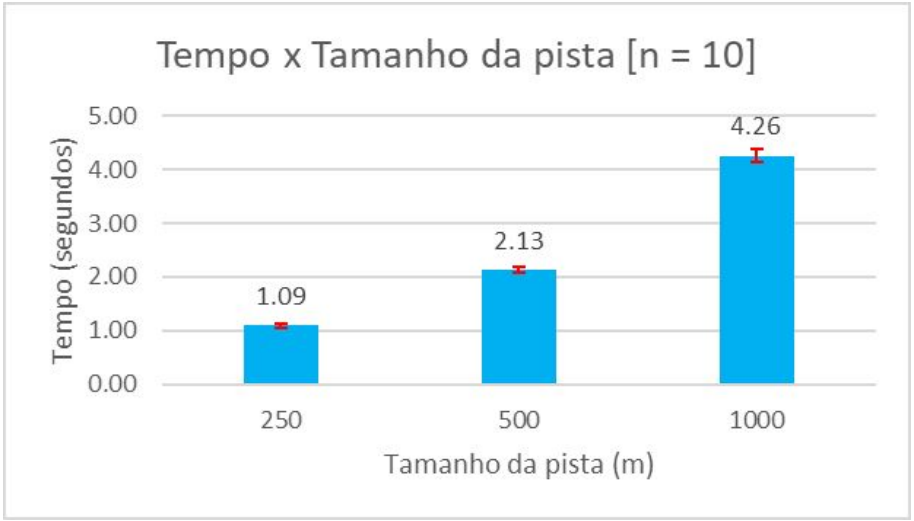


Máquina B

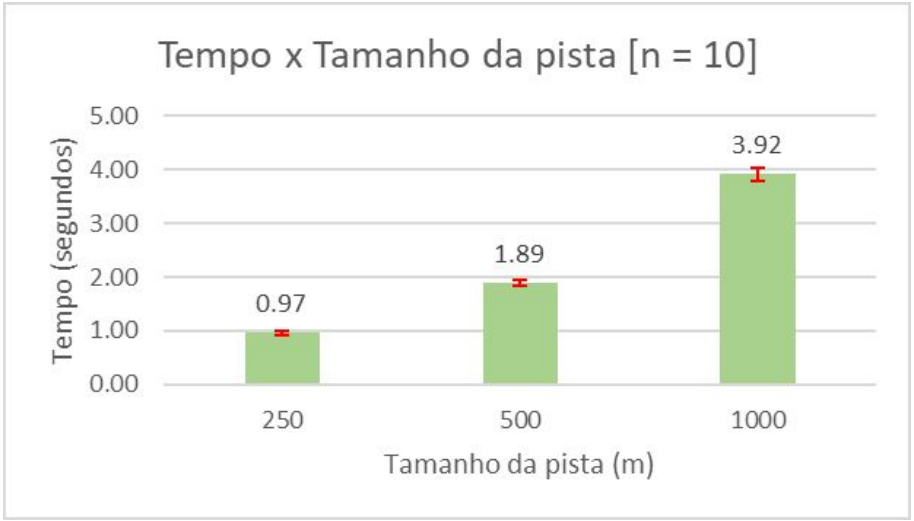
Conclusões - Gráficos - Tempo X Número de ciclistas

A relação com o aumento de ciclista mostrou-se proporcional aos respectivos aumentos de magnitude, contudo acreditamos que as distorções dessa proporcionalidade, no que diz respeito a um aumento linear (como por exemplo, os pares (10 ; 1,1) , (100 ; 9,2) e (300 ; 9,2) do primeiro gráfico) sejam devido a escolhas de tamanho de espera (sleep time) das threads esperando alguma condição acontecer.

Gráficos - Tempo x Tamanho da pista

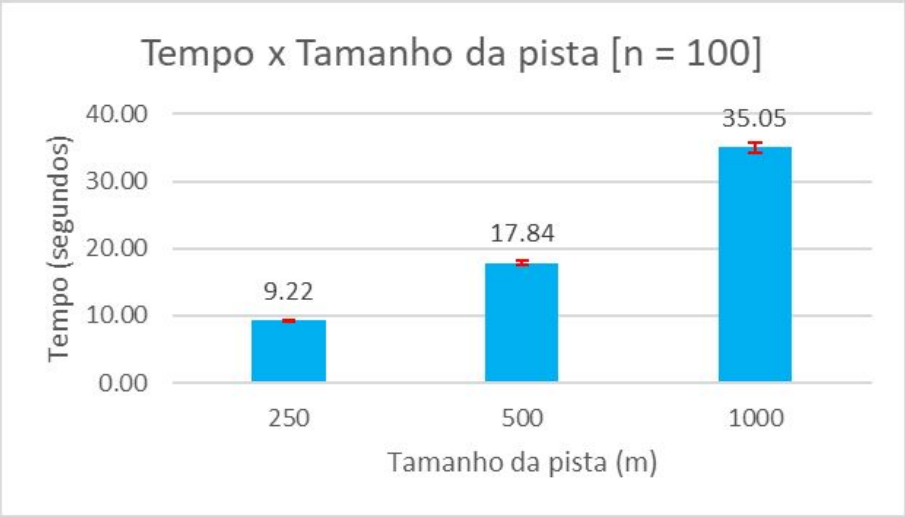


Máquina A

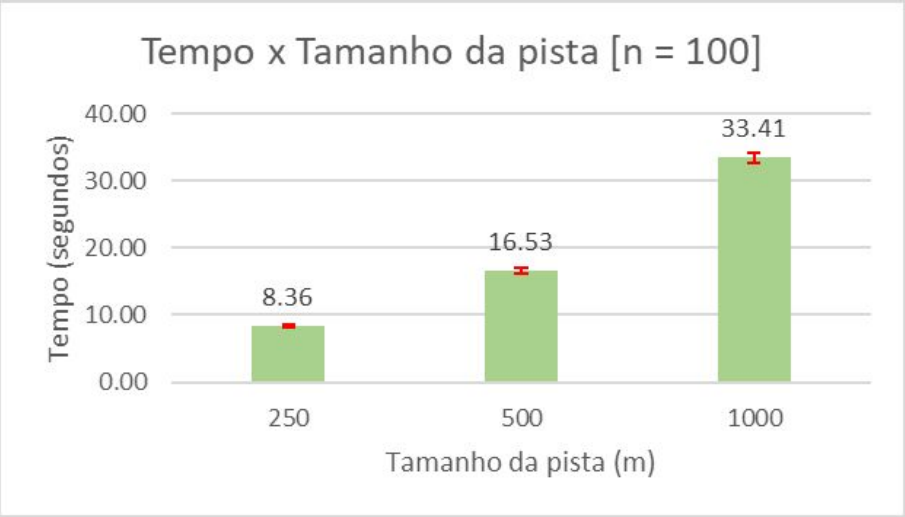


Máquina B

Gráficos - Tempo x Tamanho da pista

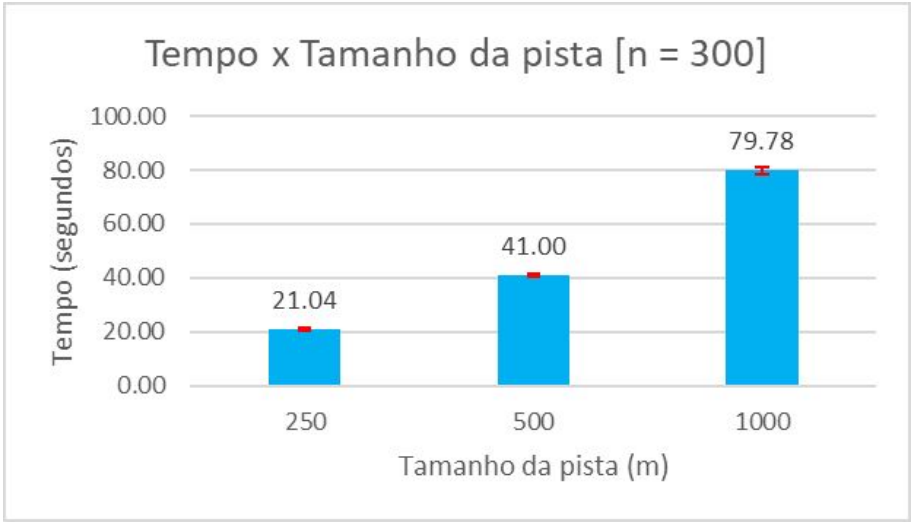


Máquina A

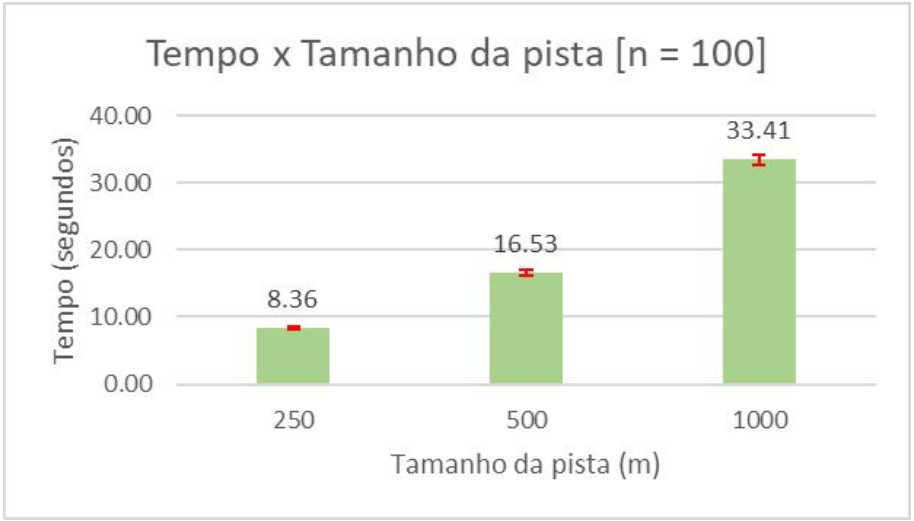


Máquina B

Gráficos - Tempo x Tamanho da pista



Máquina A

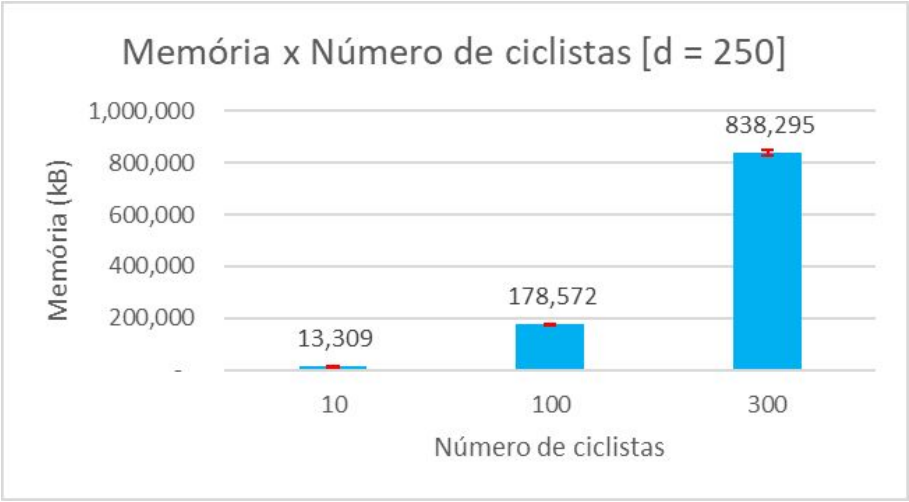


Máquina B

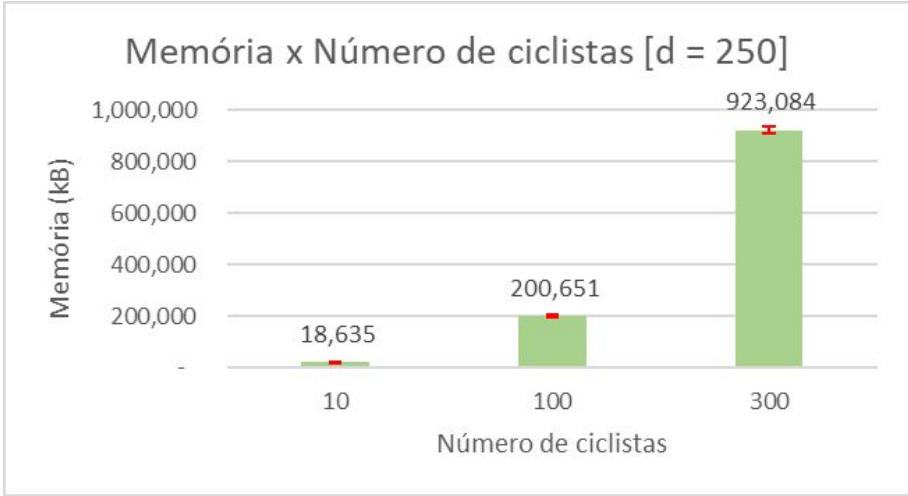
Conclusões - Tempo x Tamanho da pista

O aumento das dimensões das pistas resultaram em aumentos linearmente proporcionais, o que era esperado, e os resultados se mostraram mais consistentes quando comparado aos aumentos do caso das quantidades de ciclistas

Gráficos - Memória x Número de ciclistas

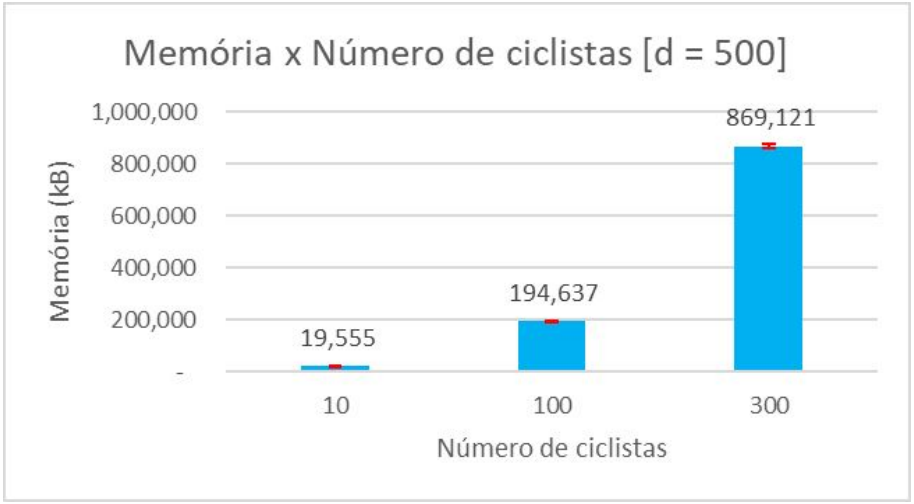


Máquina A

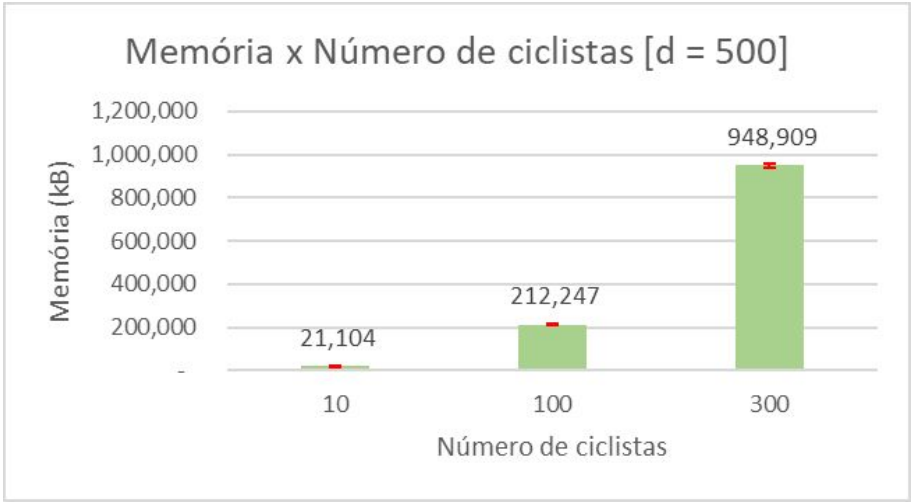


Máquina B

Gráficos - Memória x Número de ciclistas

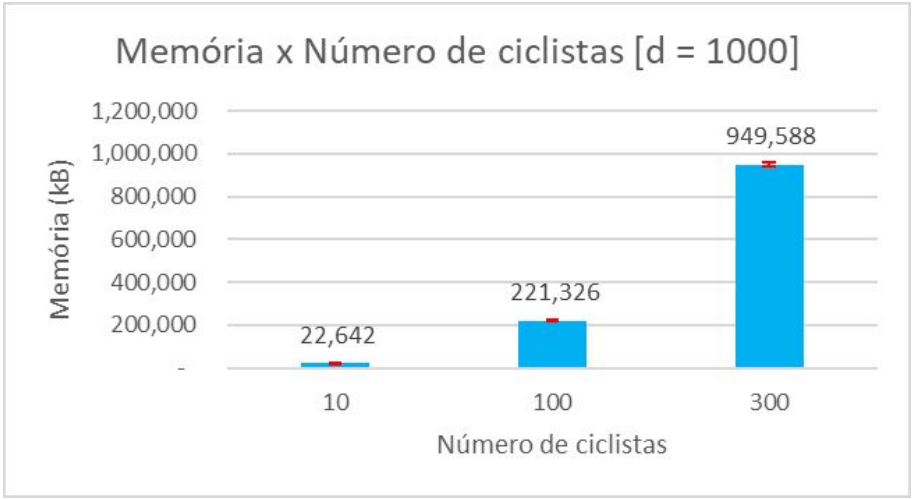


Máquina A

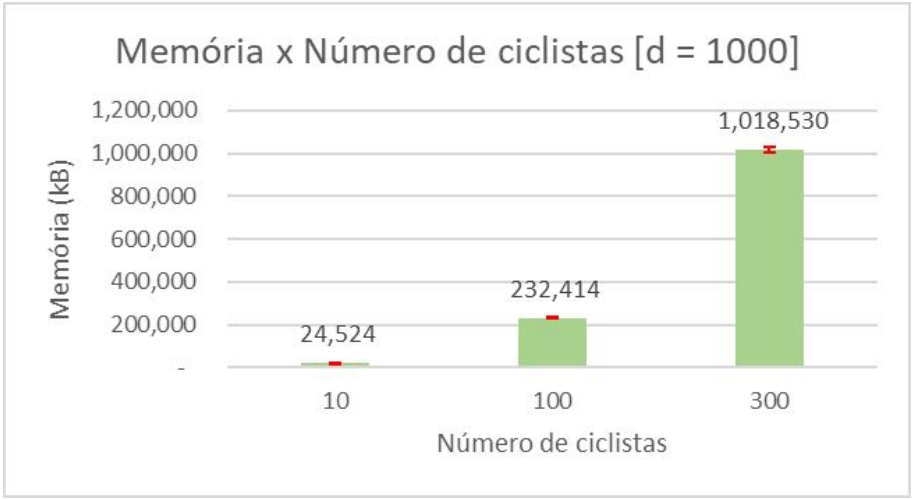


Máquina B

Gráficos - Memória x Número de ciclistas



Máquina A

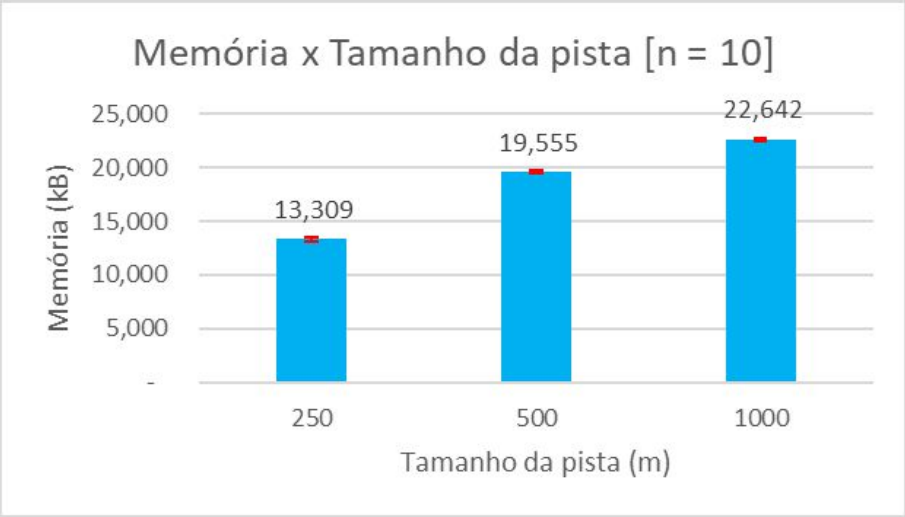


Máquina B

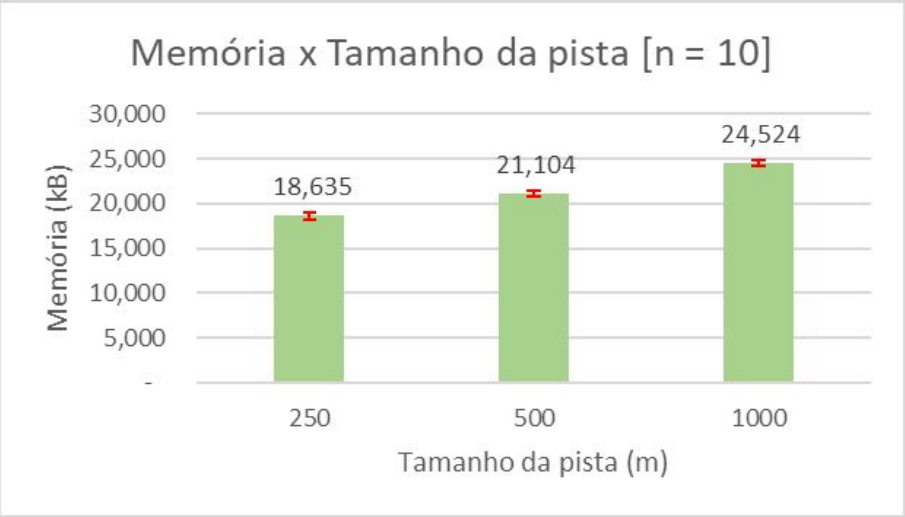
Conclusões - Memória x Número de ciclistas

O aumento do número de ciclistas tem um impacto grande no uso da memória, não somente pela criação de suas respectivas estruturas e threads, mas como também o fato de que mais threads, em relação aos núcleos lógicos do computador, resultam em uma necessidade de armazenamento de estados em memória virtual muito maior.

Gráficos - Memória x Tamanho da pista

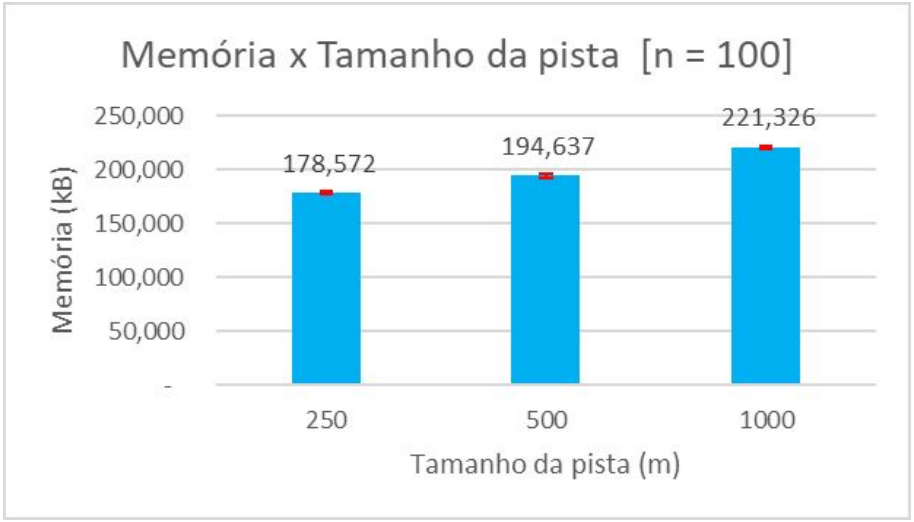


Máquina A

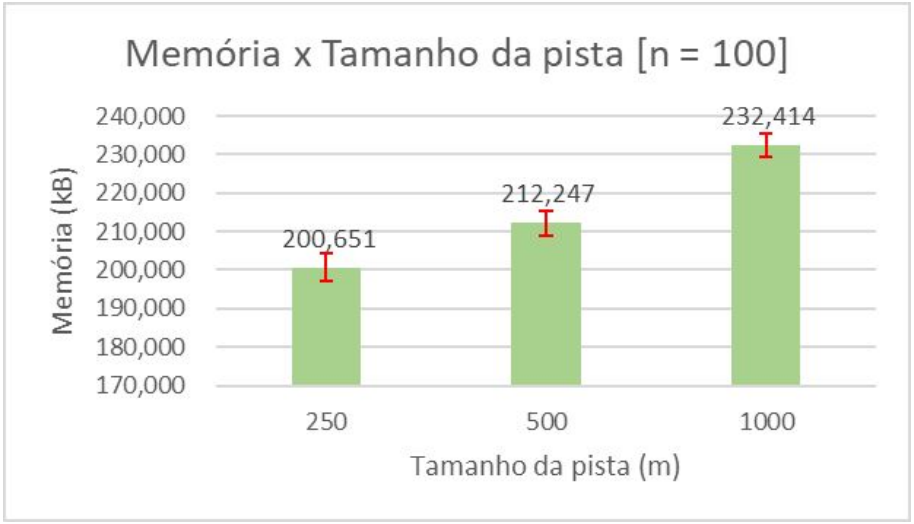


Máquina B

Gráficos - Memória x Tamanho da pista

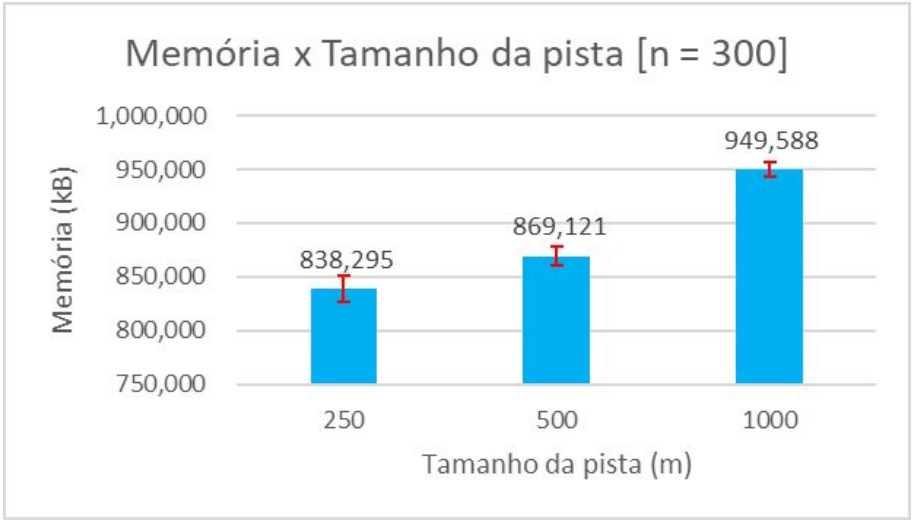


Máquina A

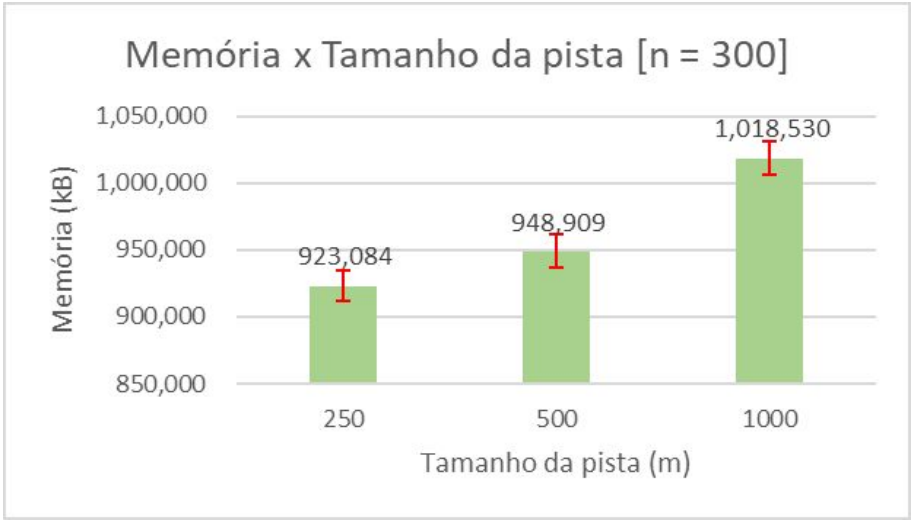


Máquina B

Gráficos - Memória x Tamanho da pista



Máquina A



Máquina B

Conclusões - Memória x Tamanho da pista

O aumento da dimensão das pistas envolveu a criação de matrizes maiores, sendo as matrizes estruturas computacionais que naturalmente consomem bastante memória, contudo esse aumento não teve a mesma expressividade que o aumento do número de ciclistas no que diz respeito ao uso da memória.

Conclusões Finais

Os requisitos deste trabalho demonstraram-se muito bons na construção do entendimento do uso dinâmico de threads e seus problemas de sincronização.

Esse cenário nos obrigou a pensar constantemente sobre condições de corrida, estrutura de dados e acesso a seção crítica.

Referências

Várias referências de uso das biblioteca pthread.h, time.h e sys/.... .h inspiradas mas não copiadas das notas de aula e sites:

- https://pt.stackoverflow.com/
- https://man7.org/
- https://linux.die.net/

Utilizamos os códigos para geração de números aleatórios usadas na disciplina MAC0328 - Algoritmos em Grafos, também existentes na seguinte página do professor Paulo Feofiloff:

- https://www.ime.usp.br/~pf/algoritmos/aulas/random.html