EP2 - MAC0422

João Gabriel e Juliano Garcia

Instituto de Matemática e Estatística - USP

Overview

1 Estruturas

2 Ciclistas

3 Gráficos

Estruturas

Estruturas de dados utilizadas

- Buffer: É um buffer estático de pontuações, indicando a colocação dos ciclistas em determinada volta;
- Pista (Road): É composta por uma matriz $dx10$ de identificadores inteiros, uma matriz $dx10$ de mutexes, e algumas variáveis de controle, como um vetor com a quantidade de ciclistas em cada faixa, variáveis que guardam as informações da entrada (número de ciclistas, tamanho da pista e número de voltas) e ponteiros para o grafo de dependência dos ciclistas e uma lista de pilhas;
- Placar (Scoreboard): É composto por uma fila estática de buffers e mais algumas variáveis de controle, como o número de ciclistas ativos e o número de ciclistas que não quebraram;

Estruturas de dados utilizadas

- Digrafo (Graph): É um grafo de dependência utilizado para nos certificarmos de que os ciclistas não irão entrar em deadlock permanente;
- Pilha (Stack): É uma pilha de inteiros utilizada em funções relacionadas ao grafo;
- Lista de pilhas (Stacklist): Uma lista de pilhas utilizada para guardar os componentes fortemente conexos do grafo;
- Ciclista (Biker): Uma estutura composta por um contador de voltas, a posição (i, j) do ciclista, seu id, sua pontuação, sua velocidade, seu tempo, sua thread, 4 mutexes, e variáveis booleanas que indicam se ele já moveu e se seus mutexes já foram usados (explicado melhor nos próximos slides).

Decisões de implementação de fluxo

- Além das threads dos ciclistas, utilizamos a thread da main para coordenar algumas operações durante a corrida, como debug e previsão de dependências ciclicas;
- Utilizamos 3 barreiras de sincronização para sincronizar todas as n+1 threads (ciclistas + main): Uma para sincronizar os movimentos dos cicistas, outra para que as threads se preparem para a próxima iteração e para imprimir o debug e a terceira para que as threads acabem ao mesmo tempo no final do programa;
- Utilizamos as barreiras disponíveis na biblioteca pthread.

Ciclistas

Decisões de implementação dos ciclistas

- Ciclistas só se movimentam para posições adjacentes à eles na matriz (incluindo diagonais).
- Ciclistas esperam os da frente para se movimentar.
- Sempre que possível, respeitando as outras regras, os ciclistas vão em direção à pista mais interna.
- Quando um ciclista quebra, criamos uma thread dummy que fica ativando as barreiras para que as outras threads continuem rodando.

Loop dos ciclistas

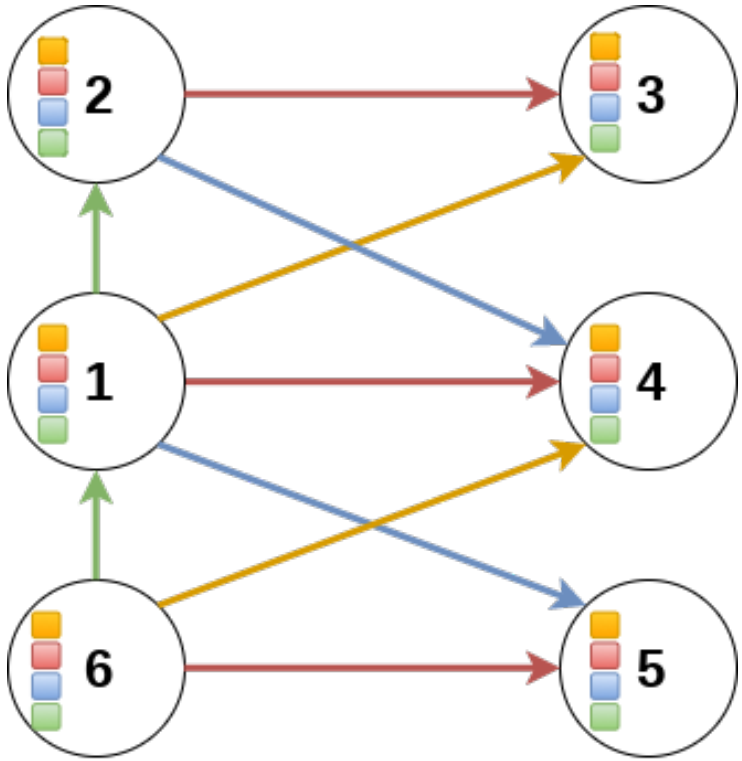
Na figura as posições que não são importantes estão representadas por "." e os ciclistas por números. O ciclista em foco é o número 1. As linhas representam as faixas e as colunas representam os metros. A faixa superior é a mais interna e os ciclistas se movem da esquerda para a direita.

Loop:

1. Espera os ciclistas que estão nas posições 2 (cima), 3 (diagonal superior direita), 4 (direita) e 5 (diagonal inferior direita) moverem para continuar.
2. Tenta se mover para a posição 3, se não conseguir tenta a 4 e se não conseguir tenta a 5.
3. Espera a barreira, e então volta para a instrução 1.

Grafo de dependência

A relação de espera para executar o movimento pode ser representada em um grafo de dependência:



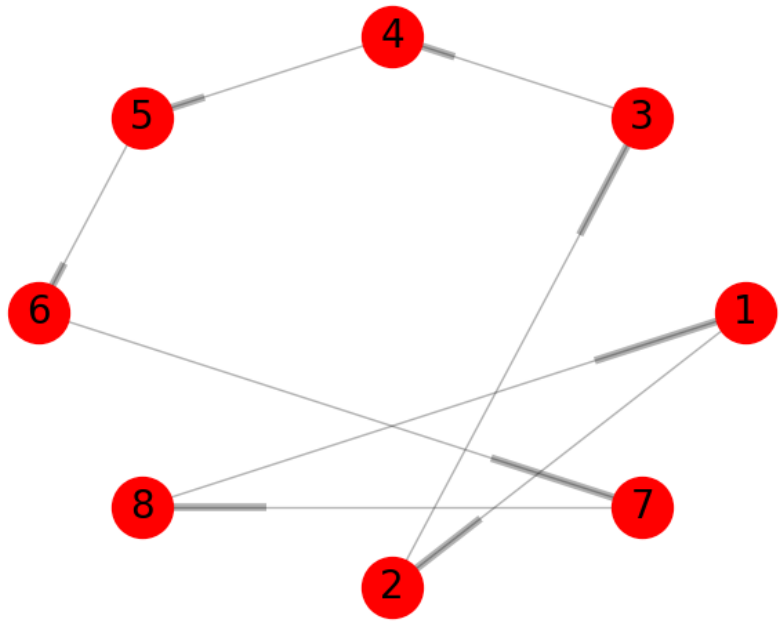
Regras extras

Para evitar alguns comportamentos indesejados, adicionamos algumas regras extras:

- Um ciclista não pode mover para a posição 3 se houver um ciclista na posição 4, evitando ultrapassagens pela pista mais interna.
- Um ciclista só pode mover para a posição 5 se o ciclista da posição 6 já se moveu, evitando que o ciclista bloqueie o ciclista da 6.
- Um ciclista não pode mudar para uma faixa que tem d-1 ciclistas, nem esperar algum ciclista que está nela, evitando que não haja espaço para movimentação.
- Se o ciclista pertence à um componente fortemente conexo (CFC) do grafo de dependência, com mais ou igual a d vértices, ele não pode mudar nem esperar faixas que possuem ciclistas de algum CFC, evitando ciclos de dependência.

Grafo de dependência e componente fortemente conexo

Sem a última regra do slide anterior, seria possível que que os ciclistas entrassem em deadlock em ciclo esperando os da frente se moverem, como é mostrado nas imagens abaixo numa pista de tamanho 8.



Grafo de dependência e componente fortemente conexo

Para isso, quando a pista tem mais de d ciclistas, o programa traduz a pista para um digrafo de dependências e acha os componentes fortemente conexos dele utilizando o **Algoritmo de Tarjan** ($\mathcal{O}(n)$).

Nosso simulador identifica todos os ciclistas que pertencem a algum CFC que tenha quantidade de nós maior ou igual ao tamanho da pista (d) e aplica a regra do slide anterior, evitando que aconteça um ciclo de deadlock.

Gráficos

Gráficos

Tamanho da pista (fixo) = 300

Quantidade de voltas:

- 20 (baixa)
- 60 (média)
- 80 (alta)

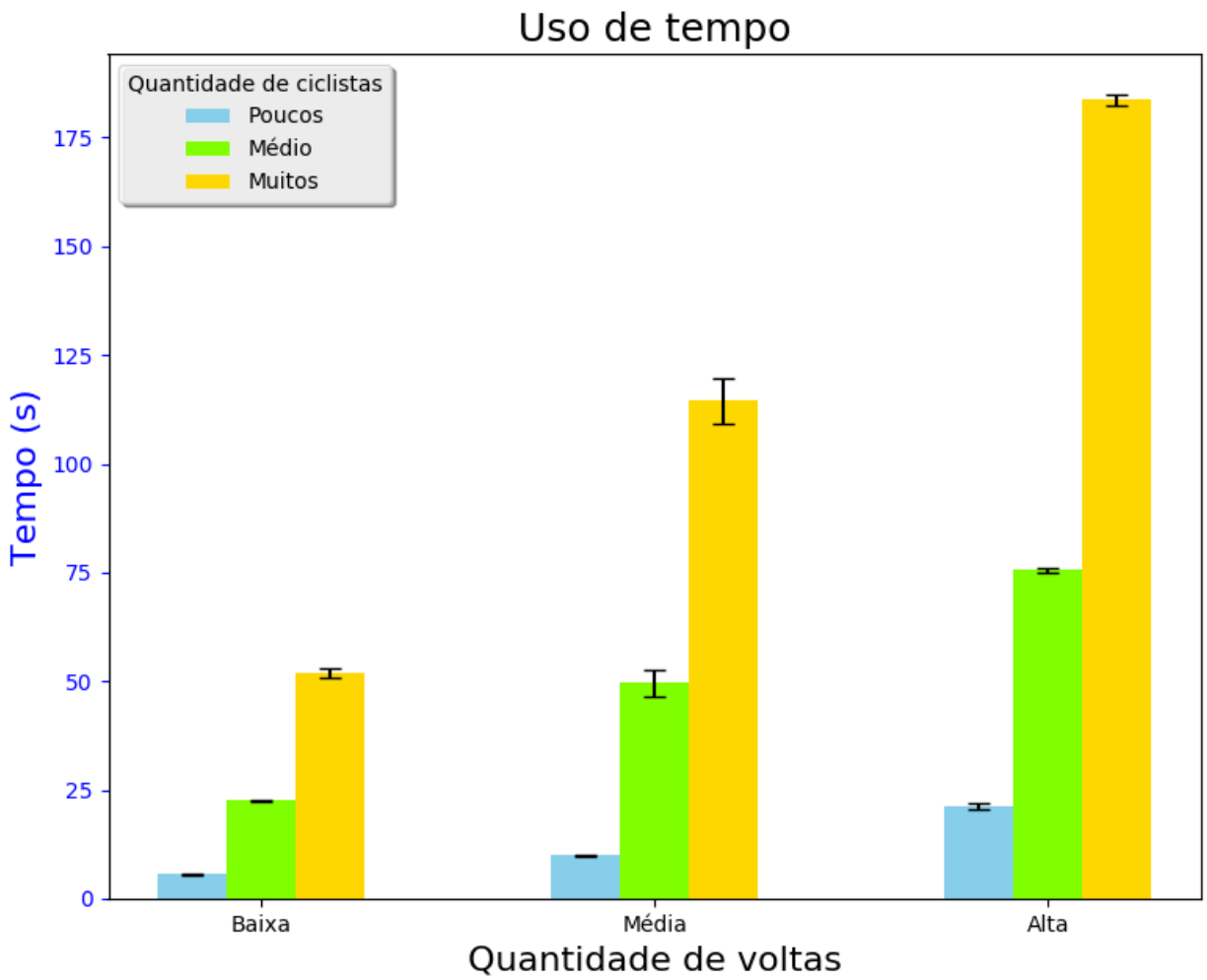
Quantidade de ciclistas:

- 40 (poucos)
- 150 (médio)
- 337 (muitos)

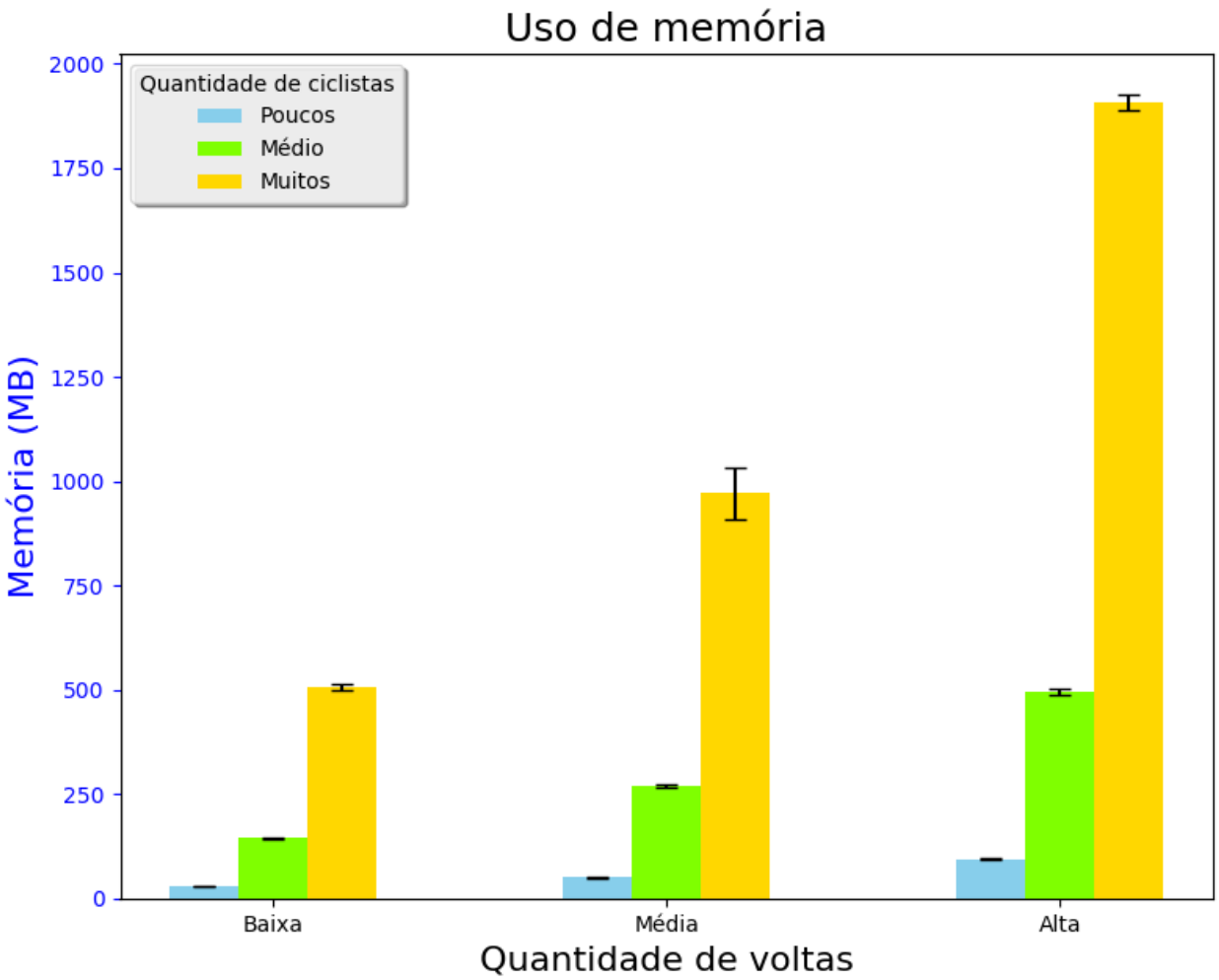
Gráficos

- O tempo foi medido usando o GNU time 1.7, com a opção 'e': "Elapsed real (wall clock) time used by the process, in seconds";
- A medição de memória foi feita utilizando o script bash **memusage**, e os dados apresentados são da seção 'heap total';
- Nos gráficos a seguir, as linhas em preto representam o intervalo de confiança, e cada barra a média de 30 medições da respectiva categoria.

Uso de tempo


Uso de tempo
Quantidade de ciclistas
Poucos
Médio
Muitos
Tempo (s)
0
25
50
75
100
125
150
175
Baixa
Média
Alta
Quantidade de voltas

Uso de memória



Conclusões

Os resultados foram os esperados:

- Quanto maior a quantidade de voltas, maior a quantidade de tempo e maior a memória gasta (mais alocações / desalocações de memória para os buffers do placar);
- Quanto maior a quantidade de ciclistas, maior o tempo de execução (mais threads para executar) e maior a memória gasta (para os buffers do placar e para o grafo de dependência).

Bibliografia

Sedgewick, Robert and Wayne, Kevin

Algorithms, 4th Edition.

Tanenbaum, Andrew S.

Modern Operating Systems, 4th Edition.