EP2 - Simulador

Aluno: Nicolas Gobbi Gonçalves – #usp: 10336188

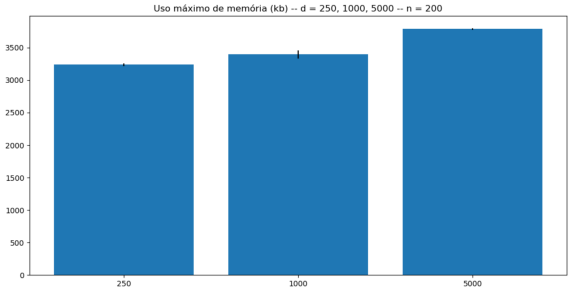
Controle de acesso a pista

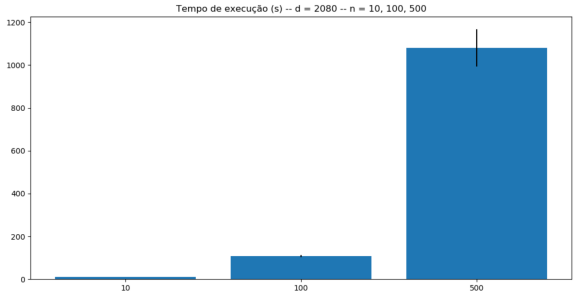
- A pista foi representada por uma matriz de inteiros de dimensão 10 x d. Para possibilitar algum paralelismo, um array de pthread_mutex foi usado, sendo que cada mutex ficou responsável por travar um metro da pista, ou seja, o array é de comprimento d.
- O interessante foi considerar o caso em que existe ao menos um corredor em cada metro da pista. Como cada corredor precisa travar dois mutex para se locomover, a situação se torna análoga ao problema dos filósofos famintos, com a possibilidade de ocorrer dead lock. Assim, para impossibilitar esse caso extremo foi usado um if...else que inverte a maneira na qual o corredor trava os mutex.

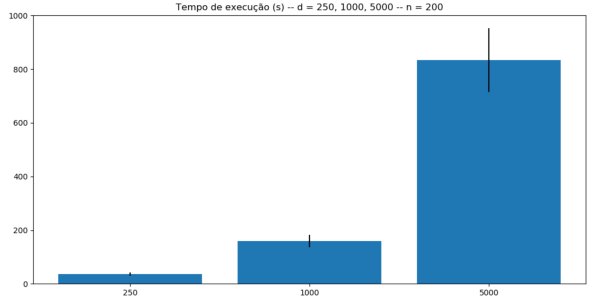
```
194     if(current_position != 0)
195       {
196         pthread_mutex_lock(&pista_locks[current_position]);
197         pthread_mutex_lock(&pista_locks[possible_next_position]);
198       }
199     else
200       {
201         pthread_mutex_lock(&pista_locks[possible_next_position]);
202         pthread_mutex_lock(&pista_locks[current_position]);
203       }
```

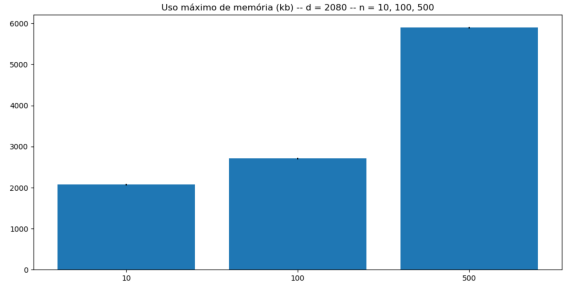
Gerenciamento dos corredores

- Os corredores foram sincronizados pelo algoritmo de worker e coordinator threads apresentado em aula. Sendo que além de servir de barreira de sincronização a thread coordenadora gerencia as threads. (Termina-as, faz o sorteio nas ultimas duas voltas, coloca-as para correr na pista).
- A quantidade de corredores para cada volta fica salva num array que é atualizado sempre que um corredor quebra.
- Em relação as colocações, um array de ponteiros para pilhas foi criado, de forma que a cada volta completada o corredor da um push na pilha referente a volta. Assim, o topo da pilha é sempre o último colocado da volta.
- Quando um corredor quebra ele vai para uma pilha de corredores quebrados, a qual é esvaziada (threads são terminadas) pela thread coordenadora sempre que está cheia.
- A velocidade é sorteada por cada corredor a cada volta completada.



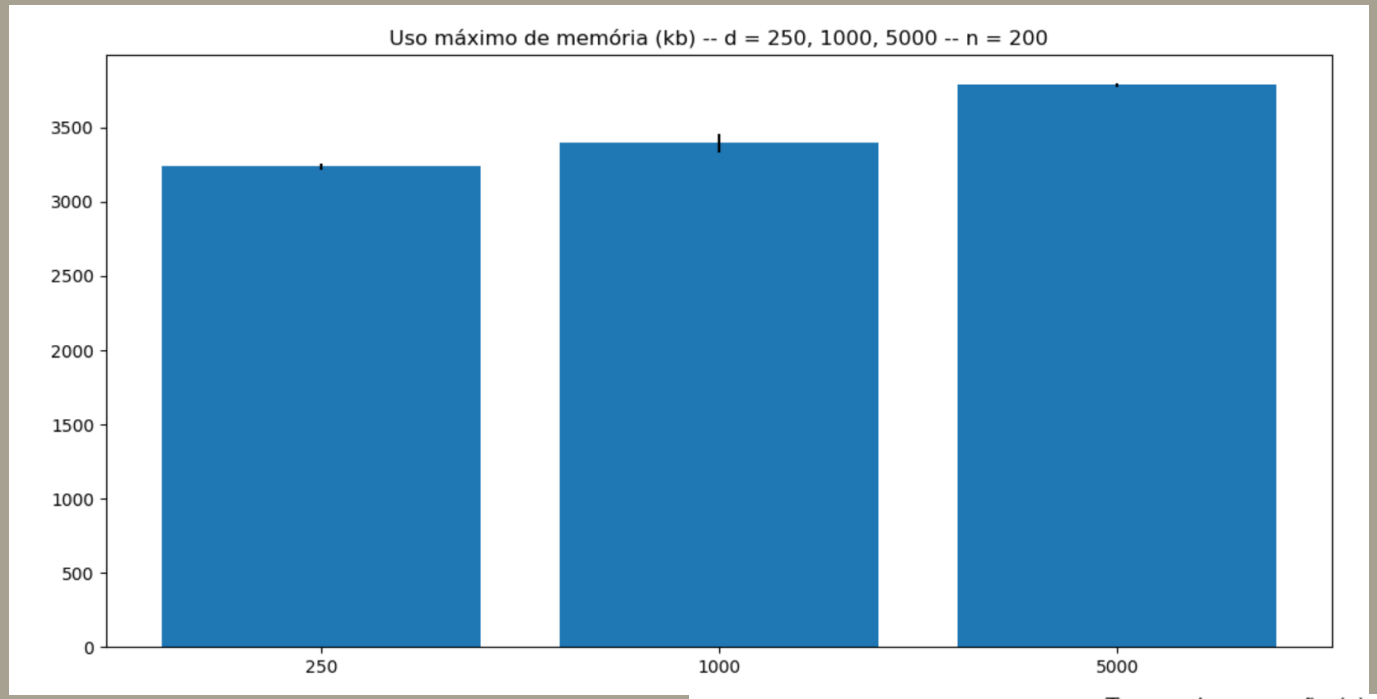




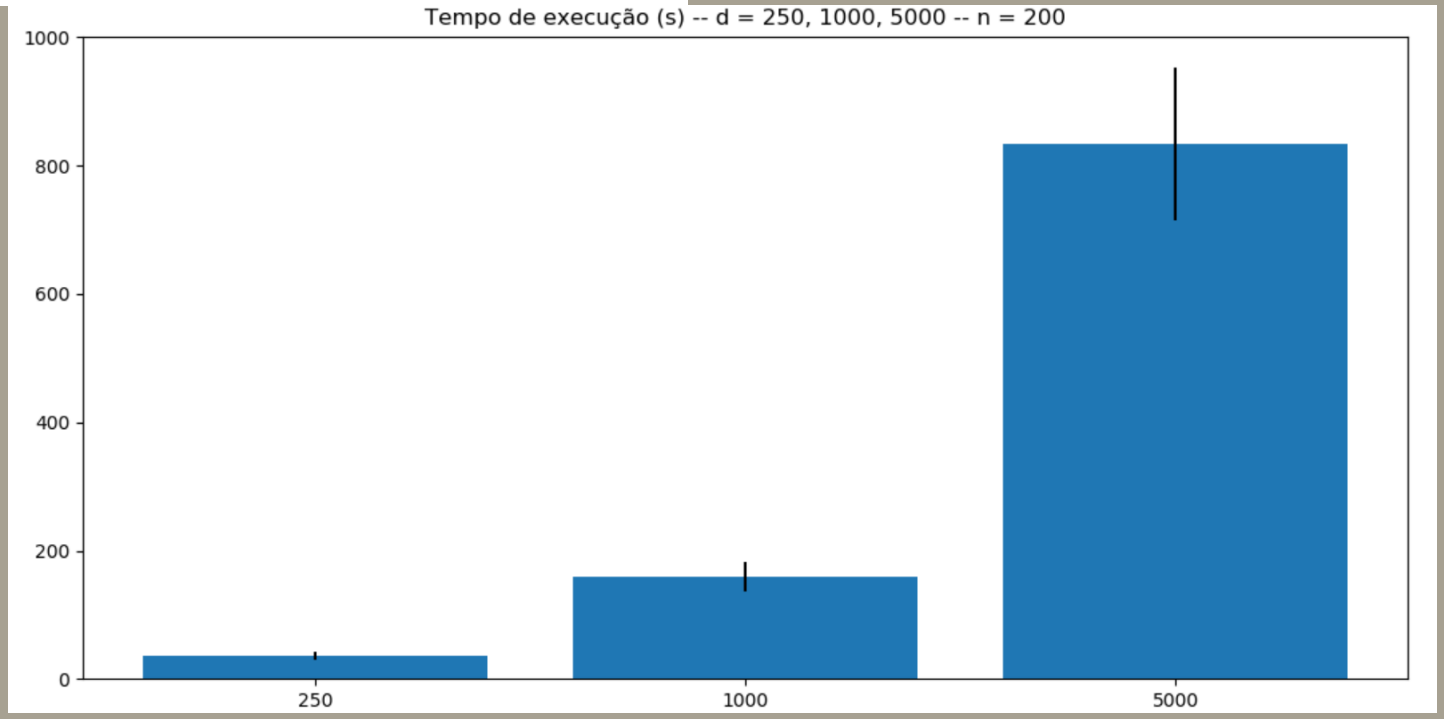


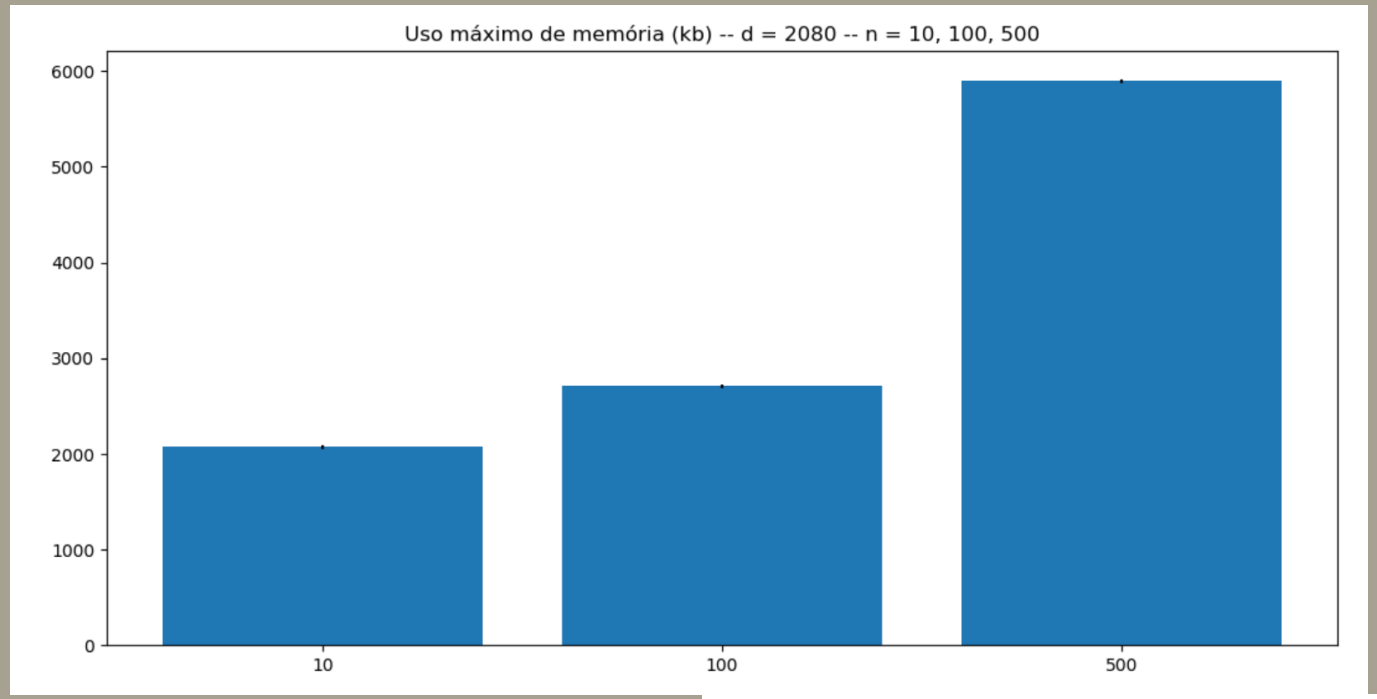
Resultados

- Testes executados num pc 4 cores
- dados obtidos rodando o comando /usr/bin/time –v ./ep2 …

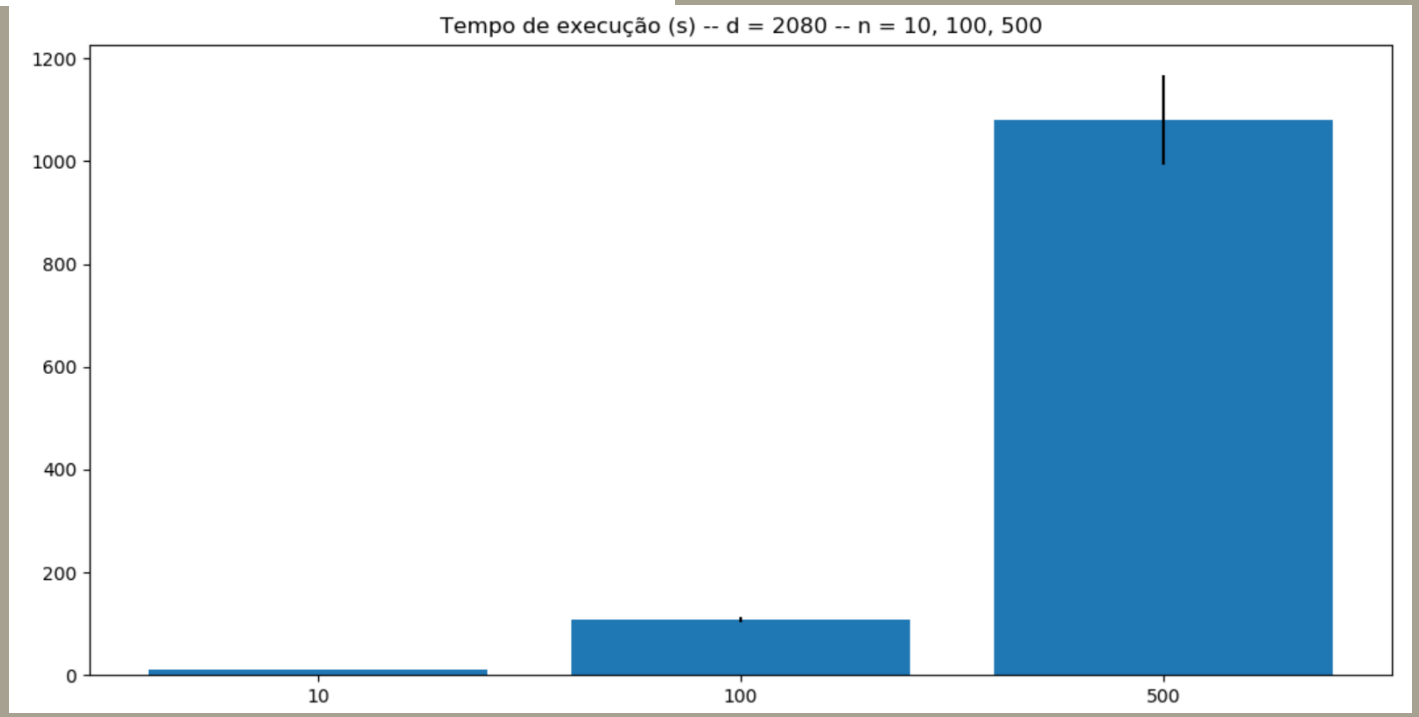

Uso máximo de memória (kb) -- d = 250, 1000, 5000 -- n = 200
3500
3000
2500
2000
1500
1000
500
0
250
1000
5000

Variando d


Tempo de execução (s) -- d = 250, 1000, 5000 -- n = 200
1000
800
600
400
200
0
250
1000
5000


Uso máximo de memória (kb) -- d = 2080 -- n = 10, 100, 500
6000
5000
4000
3000
2000
1000
0
10
100
500

Variando n


Tempo de execução (s) -- d = 2080 -- n = 10, 100, 500
1200
1000
800
600
400
200
0
10
100
500

Conclusões

- A variação no uso da memória é maior quando se varia o n, isso faz sentido:
  - Mais corredores -> mais voltas -> maior pilha de colocações
  - Mais corredores -> mais structs gestoras de corredores
  - Mais corredores -> mais threads
  - Pista maior-> só se tem aumento no tamanho do vetor pista e no vetor de mutex

- A variação no tempo de execução é grande quando se varia ambos, como era de se esperar:
  - Mais corredores -> mais threads executam por iteração -> maior tempo por iteração
  - Mais corredores -> mais voltas -> mais iterações na corrida
  - Pista maior -> mais iterações para um mesmo número de voltas