Relatório - MiniEP4
Introdução à Programação Concorrente, Paralela e Distribuída - MAC0219

Pedro Leyton Pereira, 9778794

- Especificações da máquina

A máquina no qual utilizei para fazer esse EP contém um Intel i5-6200U @ 2.7 GHz, 8GB de RAM DDR4, rodando o kernel do linux 4.9.35-1.

- Corretude do programa

Ao comentar todas as linhas contendo a função `__sync_synchronize()`, os testes falham na máquina utilizada. Descomentando as linhas novamente, os testes passam tranquilamente. Até onde consegui investigar, conclui (igual à aula) que isso se deve à instrução de atribuição não ser atômica. O 'pé na porta' não funciona caso haja estados intermediários na atribuição.

- Análise

Para fazer a análise, primeiro modifiquei o programa para que ele imprimisse um JSON em vez dos prints normais. Portanto, ao invés de linhas como

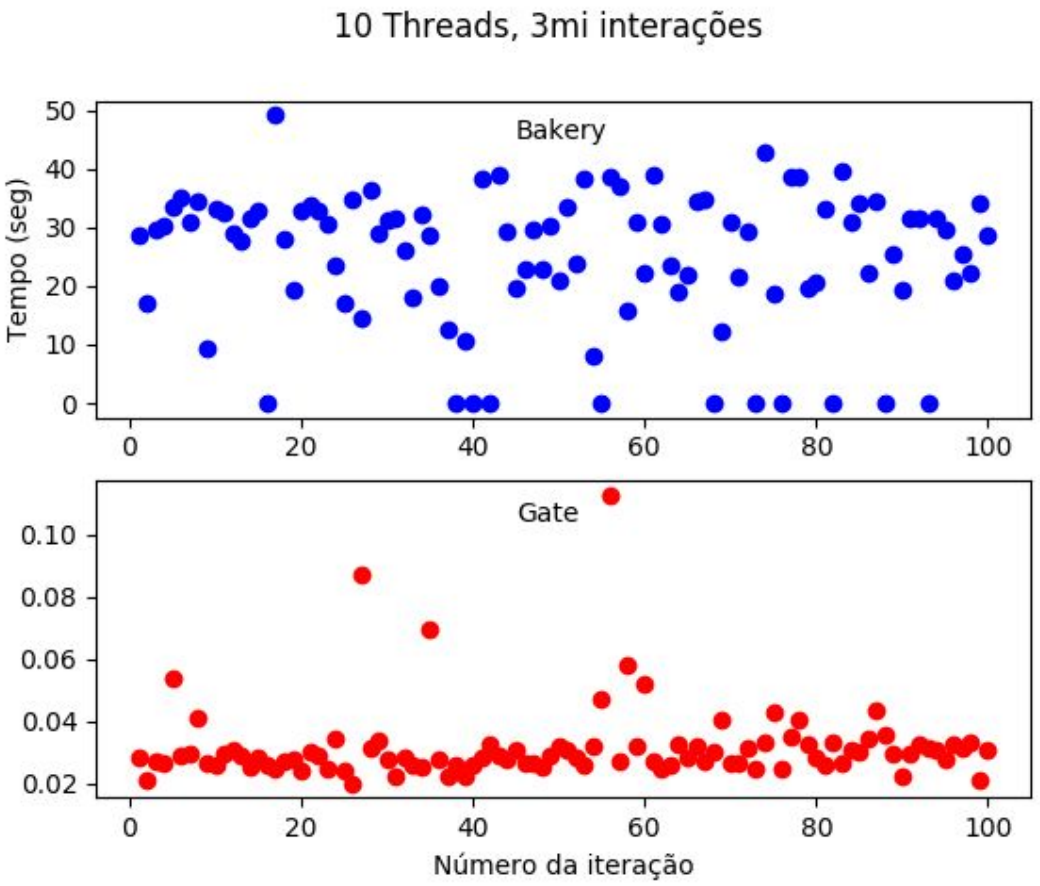
```
printf("Algorithm: %s, Execution number: %d\n", name, k+1);
```

meu programa contém linhas como

```
printf("{\n\"algorithm\": \"%s\",\n", name);
printf("\"execution_number\": %d,\n", k+1);
```

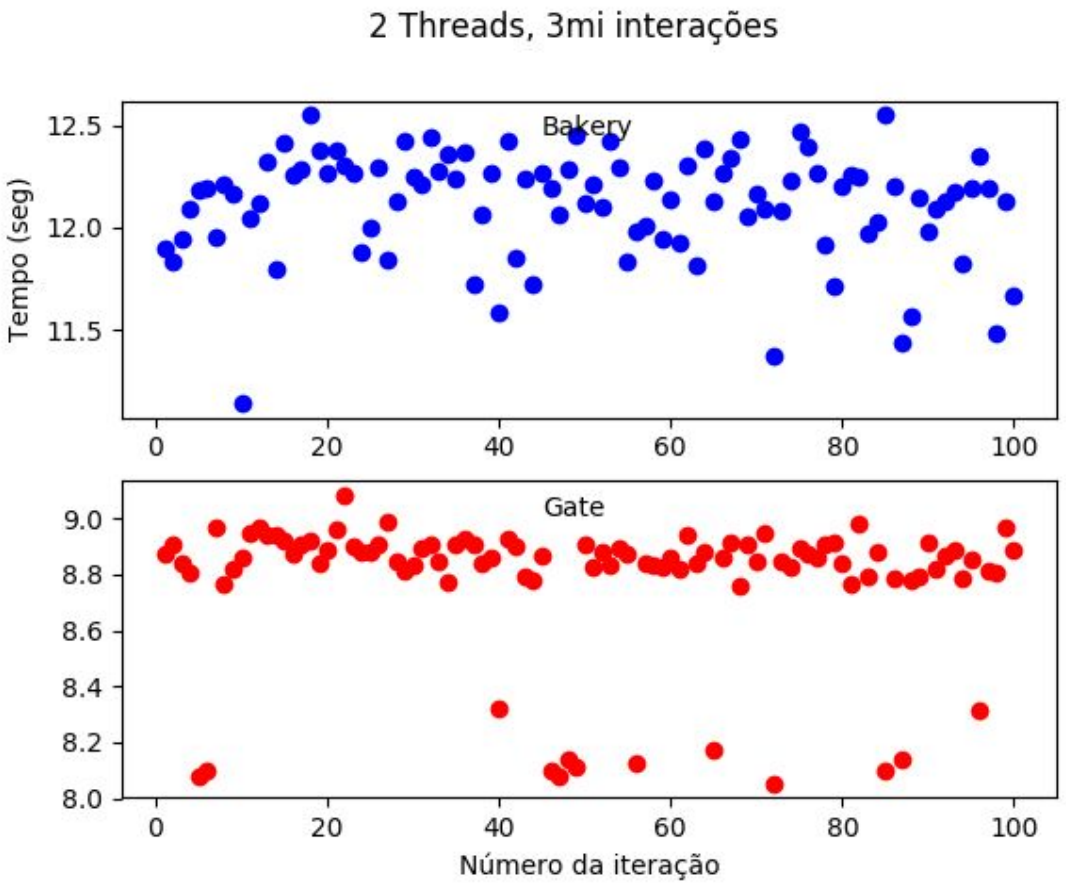
Assim, foi mais fácil rodar o programa várias vezes e importar os dados em python para fazer estatísticas e gerar gráficos com matplotlib. Além disso, aumentei o número de testes para 100 em vez do padrão de 30.

O primeiro fato que notei é de que o tempo de cada iteração varia muito para o algoritmo bakery. Para 10 threads e 3.000.000 interações com a seção crítica, o gráfico representando o tempo de cada teste ficou assim:



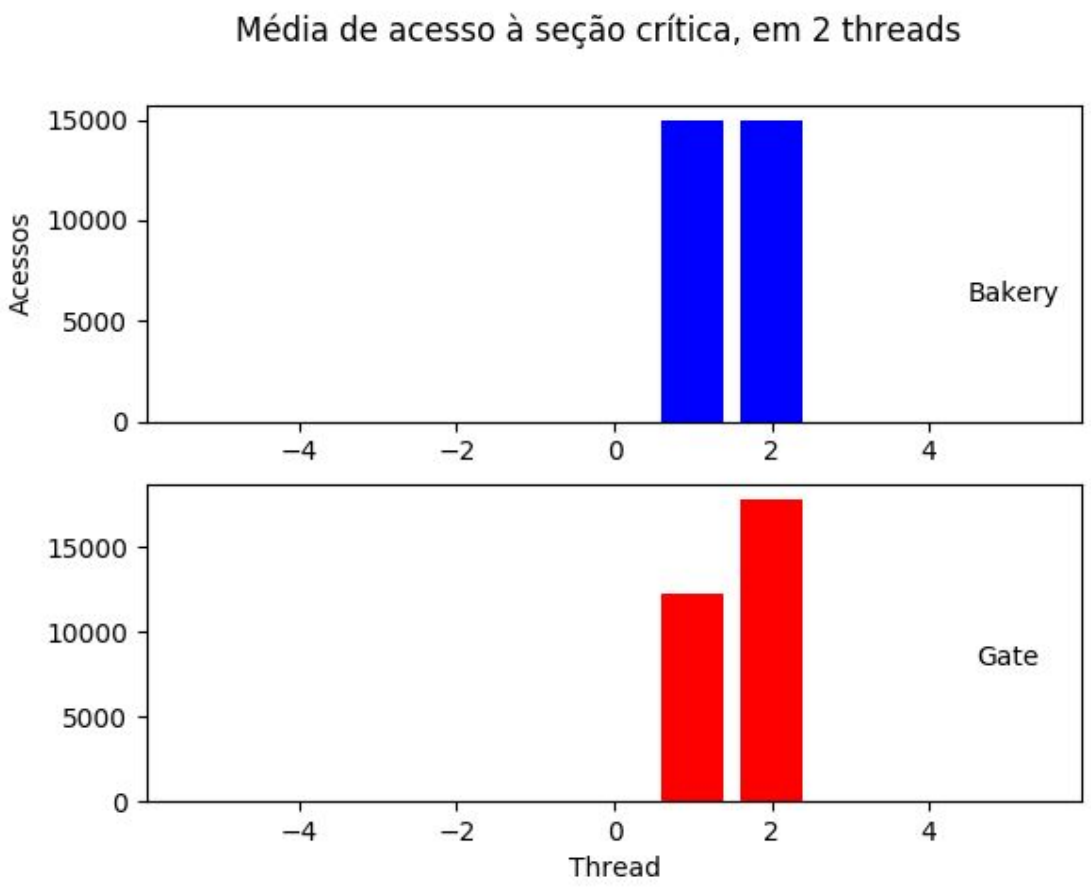

```
Bakery: Média 24.9852s, desvio padrão 115.5709s.
Gate: Média 0.0317s, desvio padrão 0.1244s.
```

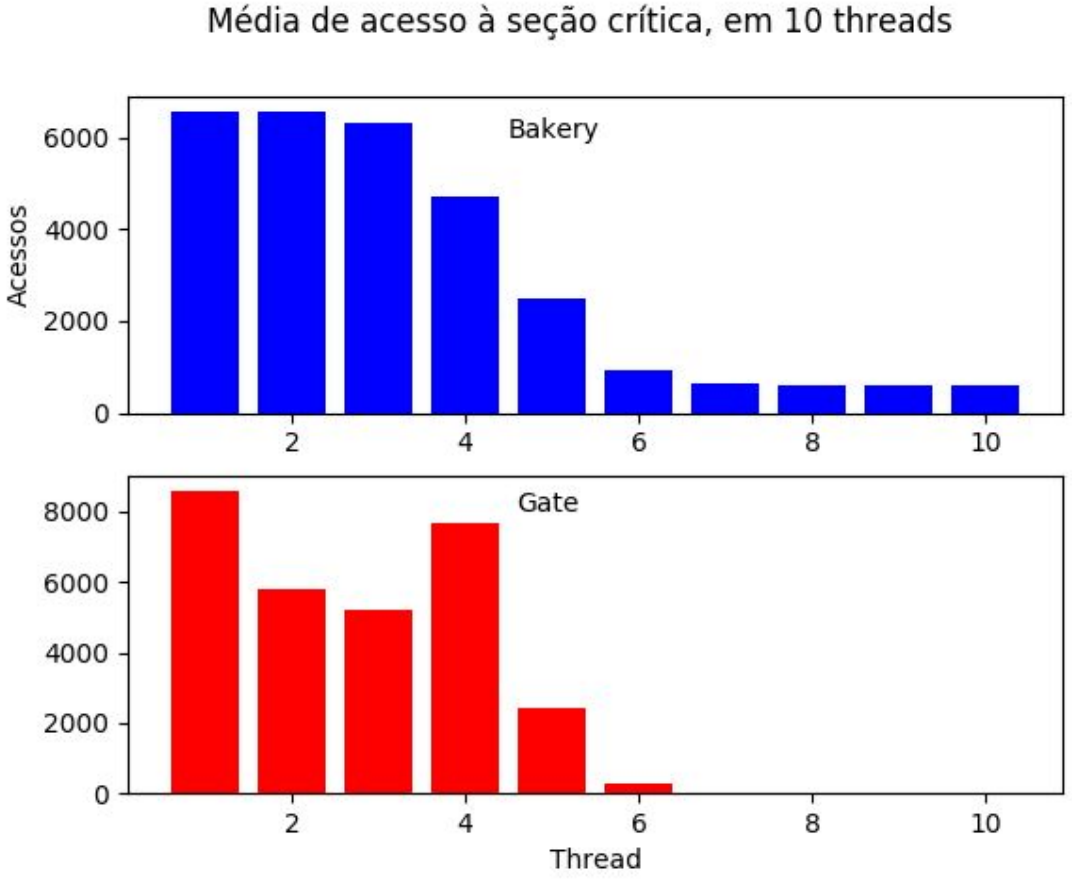
Isso indica que o algoritmo bakery implementado é muito mais inconsistente do que o algoritmo gate quando há várias threads. Com 2 threads, a história muda um pouco:




```
Bakery: Média 12.1118s, desvio padrão 2.6234s
Gate: Média 8.7764s, desvio padrão 2.5329s
```

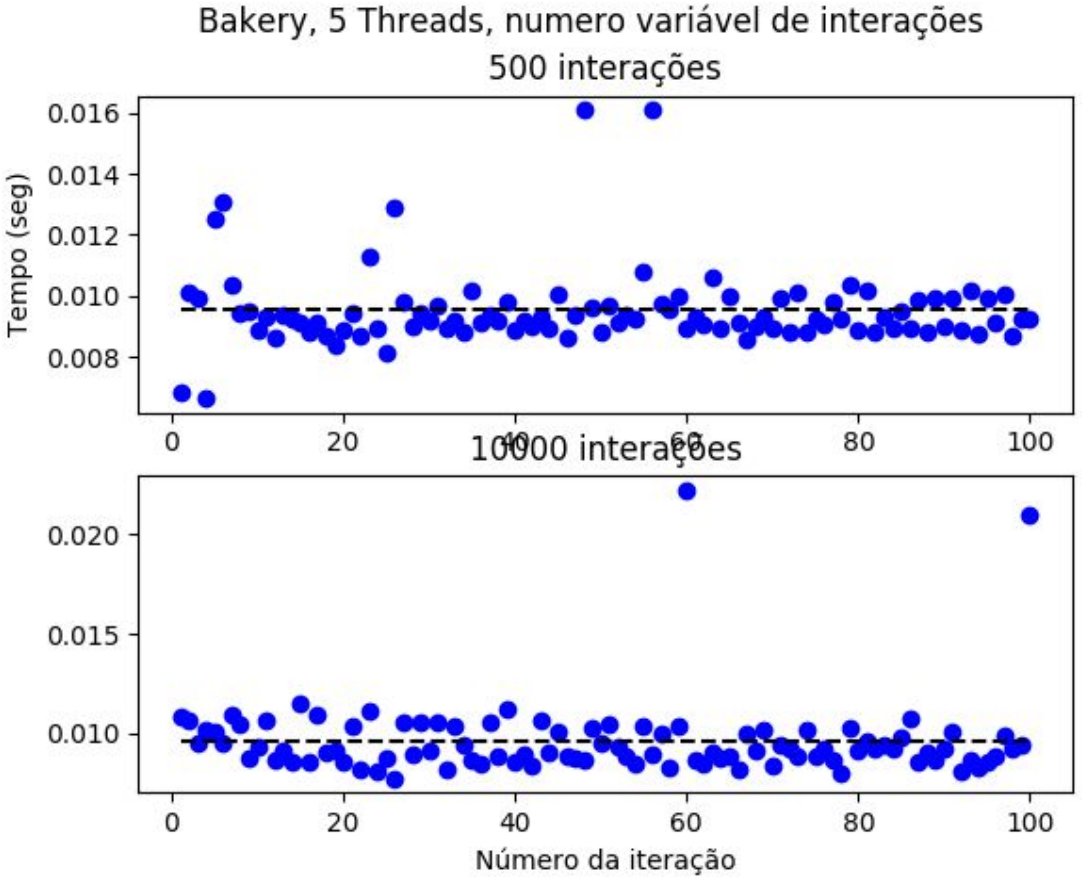
Aqui temos uma consistência muito maior no desvio padrão dos testes. Isso me deixou confuso - com mais threads, o desvio padrão decresceu muito. Fui, portanto, ver como estava a distribuição de acessos à seção crítica por threads:

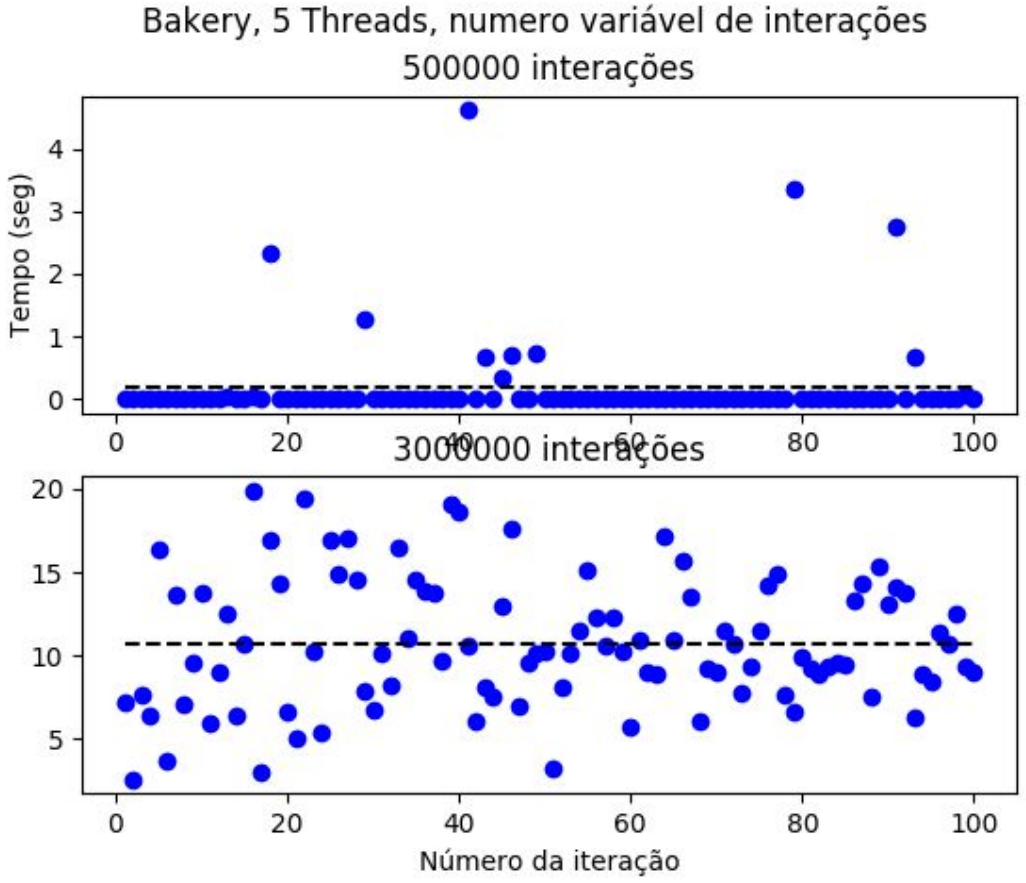

Média de acesso à seção crítica, em 2 threads
Acessos
15000
10000
5000
0
−4
−2
0
2
4
Bakery
Gate
Thread


Média de acesso à seção crítica, em 10 threads
Acessos
6000
4000
2000
0
2
4
6
8
10
Bakery
Gate
Thread

Enquanto a distribuição dos dois algoritmos em 2 threads é razoavelmente similar, a distribuição em 10 threads é completamente diferente uma da outra. Creio que a significativa diferença entre desvios padrão decorra dessa distribuição. Além disso, fica claro que o algoritmo Gate pode provocar Starvation: em 3 milhões de interações, 4 threads não acessaram nenhuma vez a seção crítica em 100 tentativas.

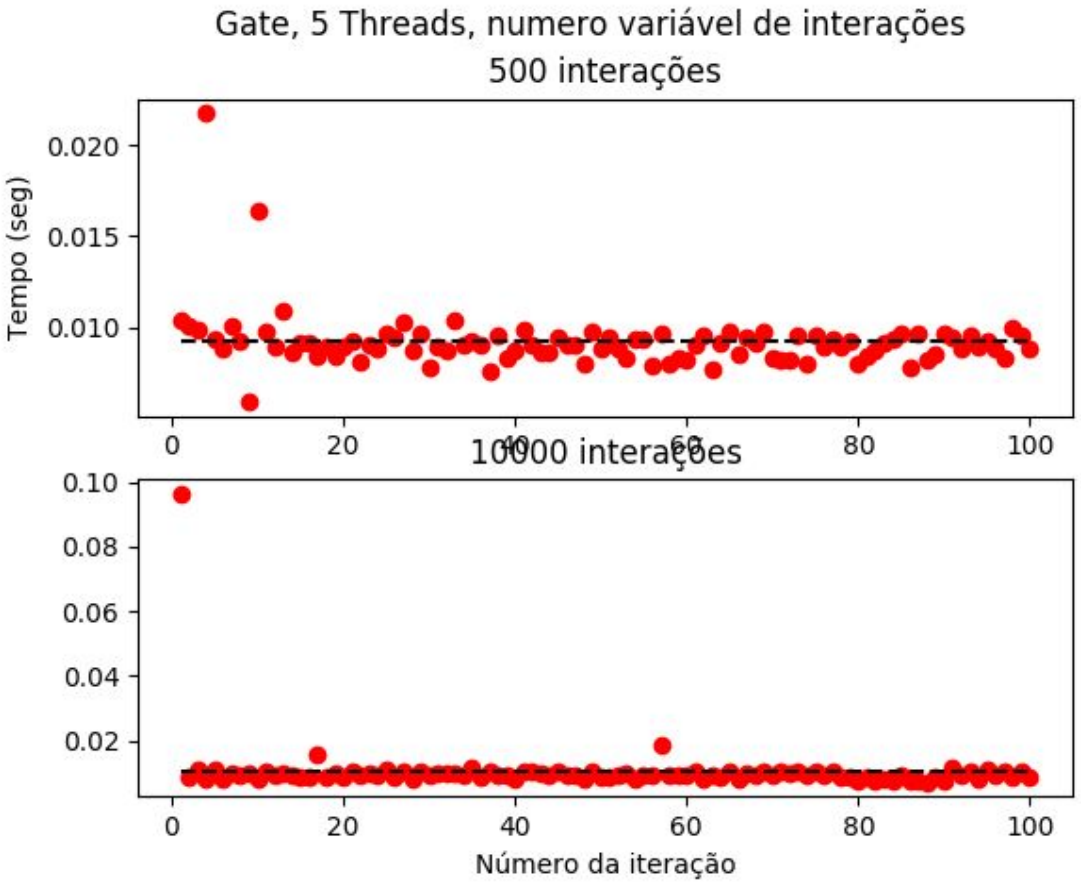
Quanto ao número de iterações, congelei o número de threads em 5, e variei o segundo parâmetro do executável (a quantidade de acessos à seção crítica).

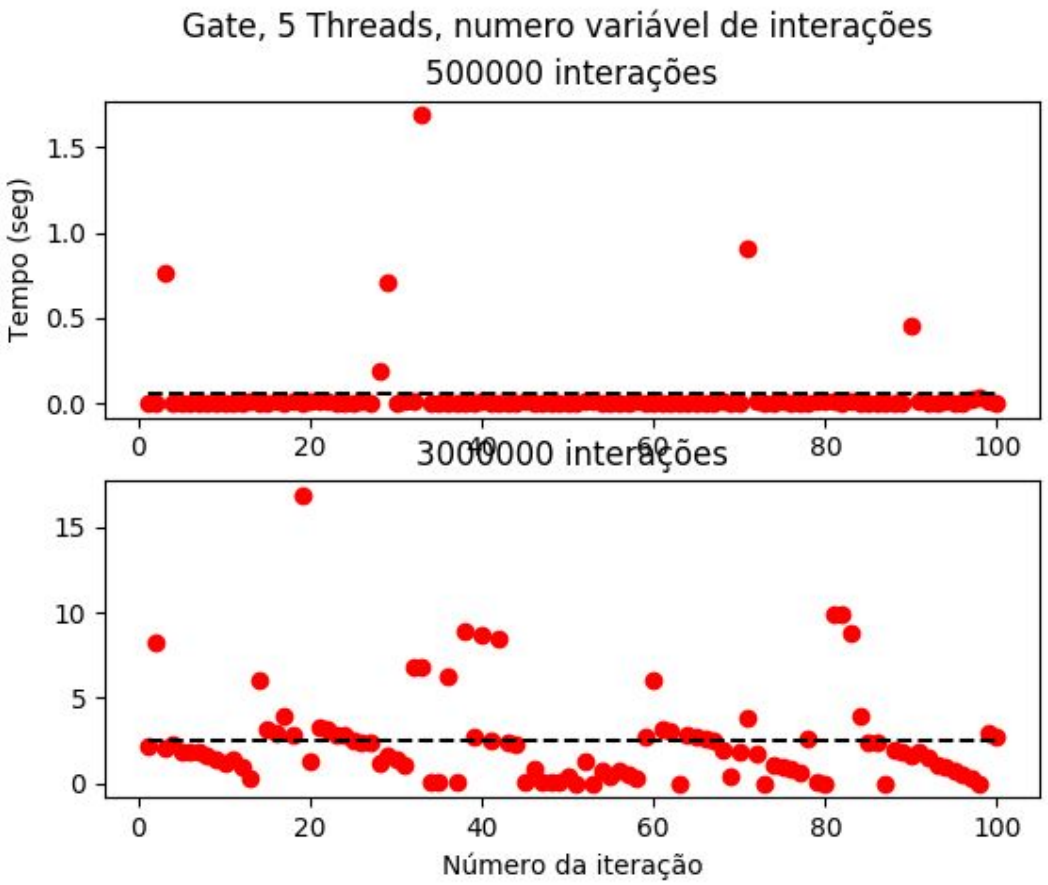

Bakery, 5 Threads, numero variável de interações
500 interações
10000 interações
Tempo (seg)
Número da iteração


Bakery, 5 Threads, numero variável de interações
500000 interações
3000000 interações
Tempo (seg)
Número da iteração

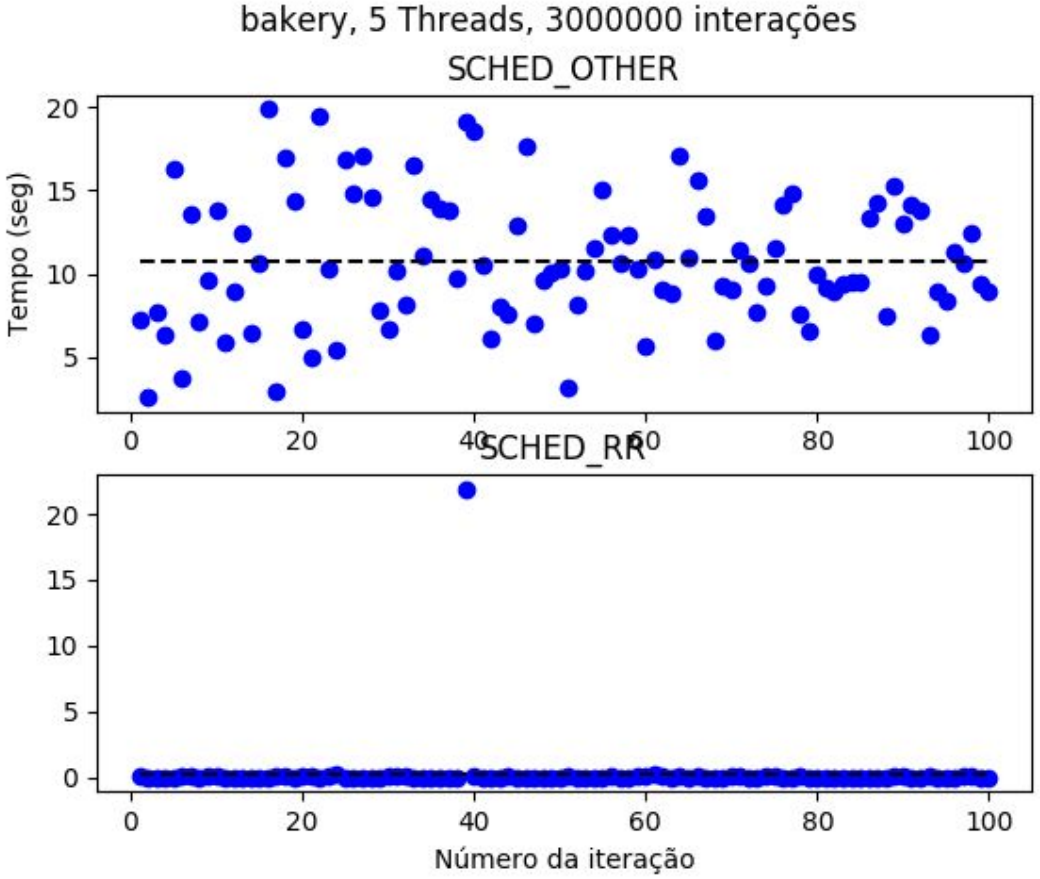
É perceptível que houve um drástico pulo na média de tempo de cada interação entre 500000 e 3000000 interações. Enquanto no primeiro a média é de 0.1843s, no segundo a média é de 10.7362s: um pulo de 58.53x, em um aumento de apenas 6x interações com a seção crítica. Comparando isso com o caso de 500 com 500000, um pulo de 1000x interações causou um pulo de apenas 19.03x no tempo de processamento.

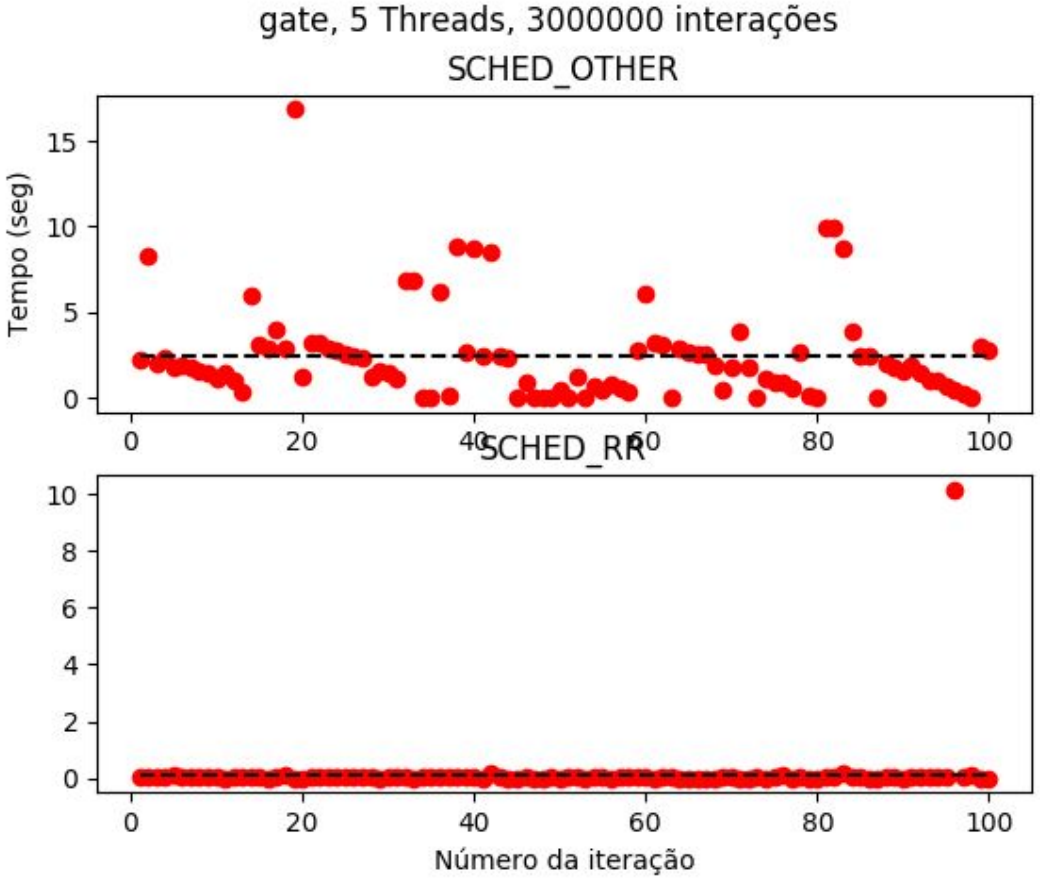
Para o algoritmo gate, a história é um pouco diferente.





Aqui ainda vemos um pulo drástico na quantidade de tempo de uma iteração ao aumentar em apenas 6x a quantidade de seções críticas. Entretanto, algo que é extremamente curioso é que apareceu um padrão no gráfico das 3000000 interações! Minha hipótese foi de que o algoritmo de escalonamento padrão do Linux deve ter influenciado bastante esse gráfico, e portanto decidi re-rodar o algoritmo com o escalonador Round Robin para ver o que aconteceria.


bakery, 5 Threads, 3000000 interações
SCHED_OTHER
Tempo (seg)
SCHED_RR
Número da iteração


gate, 5 Threads, 3000000 interações
SCHED_OTHER
Tempo (seg)
SCHED_RR
Número da iteração

Voilà. Uma simples mudança de escalonador mudou completamente o tempo da computação: não só uma média menor, mas também um desvio padrão muito mais consistente.

A conclusão é a de que o que mais influencia no tempo das iterações é o número de threads e o escalonador. Uma mudança de escalonador foi suficiente para remover o pulo gigante entre 500000 e 3000000 interações.