



Python Performance Review

Apresentando a PEP-659 e
avanços rumo a um futuro sem GIL

Fabiano Weimar [Xiru]

xiru@xiru.org





2011/07/02



de meeste mensen zwijgen, een enkeling stelt een daad

Plone



Python Performance

- Python é uma linguagem muito popular por sua simplicidade, legibilidade e bibliotecas
 - Desenvolvimento web ([Django](#), [Flask](#), [Plone](#))
 - IA, machine learning ([TensorFlow](#), [PyTorch](#), [Scikit-learn](#), [NumPy](#))
 - Ciência de dados ([Pandas](#), [Matplotlib](#))
- Usado para prototipagem e implantação de sistemas de IA em **produção**
- Grande ecossistema de desenvolvedores e comunidade ativa
- Linguagem interpretada, o que implica **desafios de desempenho**

Melhorias no Python

- A simplicidade do Python implica um custo de performance devido à sua natureza interpretada
- Processamento paralelo e gerenciamento de memória são complexos
- Versões modernas do Python tem recebido melhorias relevantes de performance
 - [PEP-659](#): Specializing Adaptive Interpreter
 - [PEP 684](#): A Per-Interpreter GIL
 - [PEP-703](#): Making the Global Interpreter Lock Optional in CPython
 - [PEP-744](#): JIT Compilation

PEP-659: Especialização Adaptativa do Interpretador

- Aumenta o desempenho do Python ao especializar o código em tempo de execução, baseado nos dados utilizados em caminhos de código frequentemente executados
- Reduz a necessidade de otimização manual
- Não exige que o desenvolvedor modifique código
- Justifica o esforço de upgrade para o [Python 3.11](#)
 - versão 3.12 é ainda melhor! (na maioria das vezes)

Operação	Forma	Especialização	Aceleração da operação	Contribuidor(es)
Operações binárias	<code>x + x</code> <code>x - x</code> <code>x * x</code>	Adicionar, multiplicar e subtrair binários para tipos comuns como int , float e str usam caminhos rápidos personalizados para seus tipos subjacentes.	10%	Mark Shannon, Donghee Na, Brandt Bucher, Dennis Sweeney
Subscrição	<code>a[i]</code>	<p>Subscrever tipos contêineres como list, tuple e dict indexam diretamente as estruturas de dados subjacentes.</p> <p>Subscrever __getitem__() personalizado também é inserido em linha de forma similar a Chamadas de função Python em linha.</p>	10-25%	Irit Katriel, Mark Shannon
Armazenar com subscrição	<code>a[i] = z</code>	Similar à especialização de subscrição acima.	10-25%	Dennis Sweeney

Chamadas	<code>f(arg)</code> <code>C(arg)</code>	Chamadas para funções e tipos (C) embutidos comuns como <code>len()</code> e <code>str</code> chamam diretamente sua versão C subjacente. Isso evita passar pela convenção de chamada interna.	20%	Mark Shannon, Ken Jin
Carregar variável global	<code>print</code> <code>len</code>	O índice do objeto no espaço de nomes globais/embutidas é armazenado em cache. Carregar globais e embutidas requer zero pesquisas de espaço de nomes.	[1]	Mark Shannon
Carregar atributo	<code>o.attr</code>	Semelhante ao carregamento de variáveis globais. O índice do atributo dentro do espaço de nomes da classe/objeto é armazenado em cache. Na maioria dos casos, o carregamento de atributos exigirá zero pesquisas de espaço de nomes.	[2]	Mark Shannon
Carregar métodos para chamada	<code>o.meth()</code>	O endereço real do método é armazenado em cache. O carregamento de método agora não tem pesquisas de espaço de nomes – mesmo para classes com longas cadeias de herança.	10-20%	Ken Jin, Mark Shannon

Armaze- nar atributo	<code>o.attr = z</code>	Semelhante à otimização de carregamento de atributos.	2% no pyperformance	Mark Shannon
Desempa- cotar sequência	<code>*seq</code>	Especializado para contêineres comuns como list e tuple . Evita convenção de chamada interna.	8%	Brandt Bucher

PEP-703: Global Interpreter Lock Optional in CPython

aka “free threading Python”

“Quando Python se livrar do GIL, o uso de threads permitirá que código seja executado concorrentemente de forma eficiente”

- muitos pythonistas (otimistas)



Ajustando o kernel scheduler

```
$ sudo su -
```

```
# for a in $(seq 0 15); { echo performance >  
/sys/devices/system/cpu/cpu$a/cpufreq/scaling_governor; }
```

```
$ cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor | uniq  
performance
```

n-queens

Generalização do problema das 8 rainhas

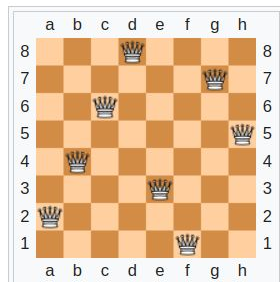
(https://en.wikipedia.org/wiki/Eight_queens_puzzle)

com um número N de rainhas

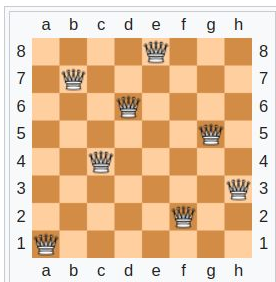
Avaliemos como diferentes versões de Python se comportam, exercitando as melhorias das PEP-659 (especialização adaptativa), PEP-703 (free threading) e PEP-744 (experimental jit), etc.

Os [testes](#) serão executados em um laptop AMD [Ryzen 7 4800HS](#) com **16 threads** (o código “*usa apenas*” 10 threads). Vamos repetir os testes usando um servidor Intel Dual [Xeon E5-2683 v4](#) com **64 threads**.

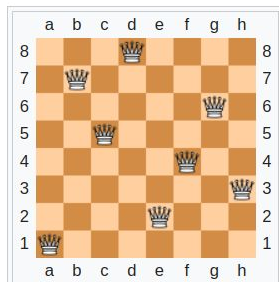
n	soluções
1	1
2	0
3	0
4	2
5	10
6	4
7	40
8	92
9	352
10	724
11	268
12	142
13	73712
14	365596
15	2279184
16	14772512
17	95815104
18	666090624
19	4968057848
20	39029188884



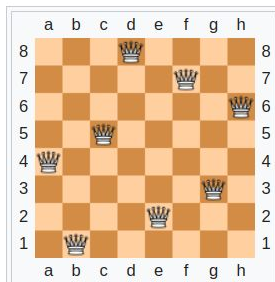
Solution 1



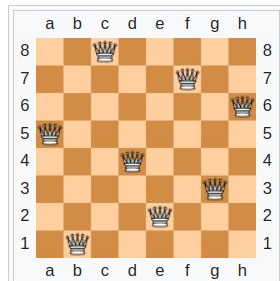
Solution 2



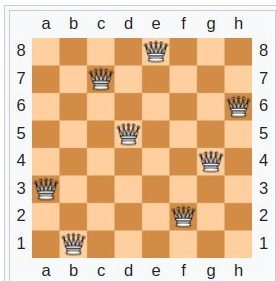
Solution 3



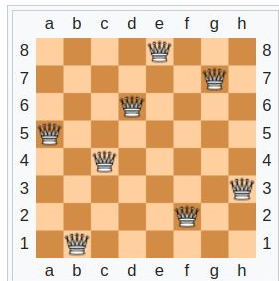
Solution 4



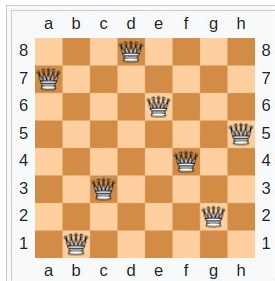
Solution 5



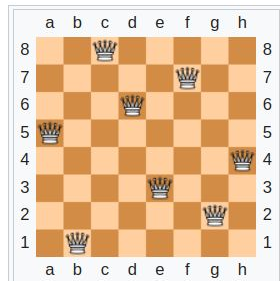
Solution 6



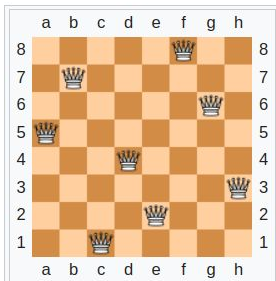
Solution 7



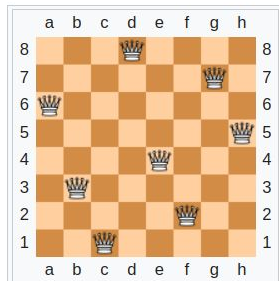
Solution 8



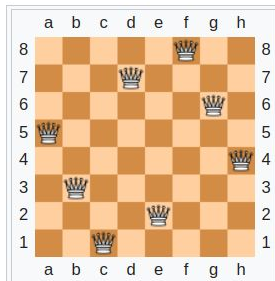
Solution 9



Solution 10



Solution 11



Solution 12

```

from utils import timeit

N_QUEENS_SIZE = 12

class NQueens:
    def solve(self, n):
        col = set()
        posDiag = set()
        negDiag = set()

        res = []
        board = [["."] * n for i in range(n)]

        def backtrack(r=0):
            if r == n:
                res.append("\n".join(["".join(row) for row in
board]))

                return

            for c in range(n):
                if c in col or (r + c) in posDiag or (r - c) in
negDiag:

                    continue

                col.add(c)
                posDiag.add(r + c)
                negDiag.add(r - c)
                board[r][c] = "Q"

                backtrack(r + 1)

```

```

        col.remove(c)
        posDiag.remove(r + c)
        negDiag.remove(r - c)
        board[r][c] = "."

        backtrack()

        return res

def solve():
    NQueens().solve(N_QUEENS_SIZE)

@timeit
def run():
    for _ in range(10):
        solve()

@timeit
def run_threads():
    from concurrent.futures.thread import ThreadPoolExecutor
    with ThreadPoolExecutor(max_workers=10) as executor:
        for _ in range(10):
            executor.submit(solve)

run()
run_threads()

```

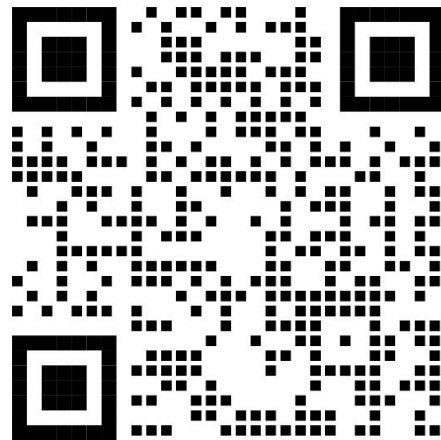
```
from utils import timeit

# https://discuss.python.org/t/about-free-threading-performance/54604/1
def fib(n):
    if n < 3:
        return n
    return fib(n-1) + fib(n-2)

@timeit
def run():
    for _ in range(10):
        fib(35)

@timeit
def run_threads():
    from concurrent.futures.thread import ThreadPoolExecutor
    with ThreadPoolExecutor(max_workers=10) as executor:
        for _ in range(10):
            executor.submit(fib, 35)

run()
run_threads()
```

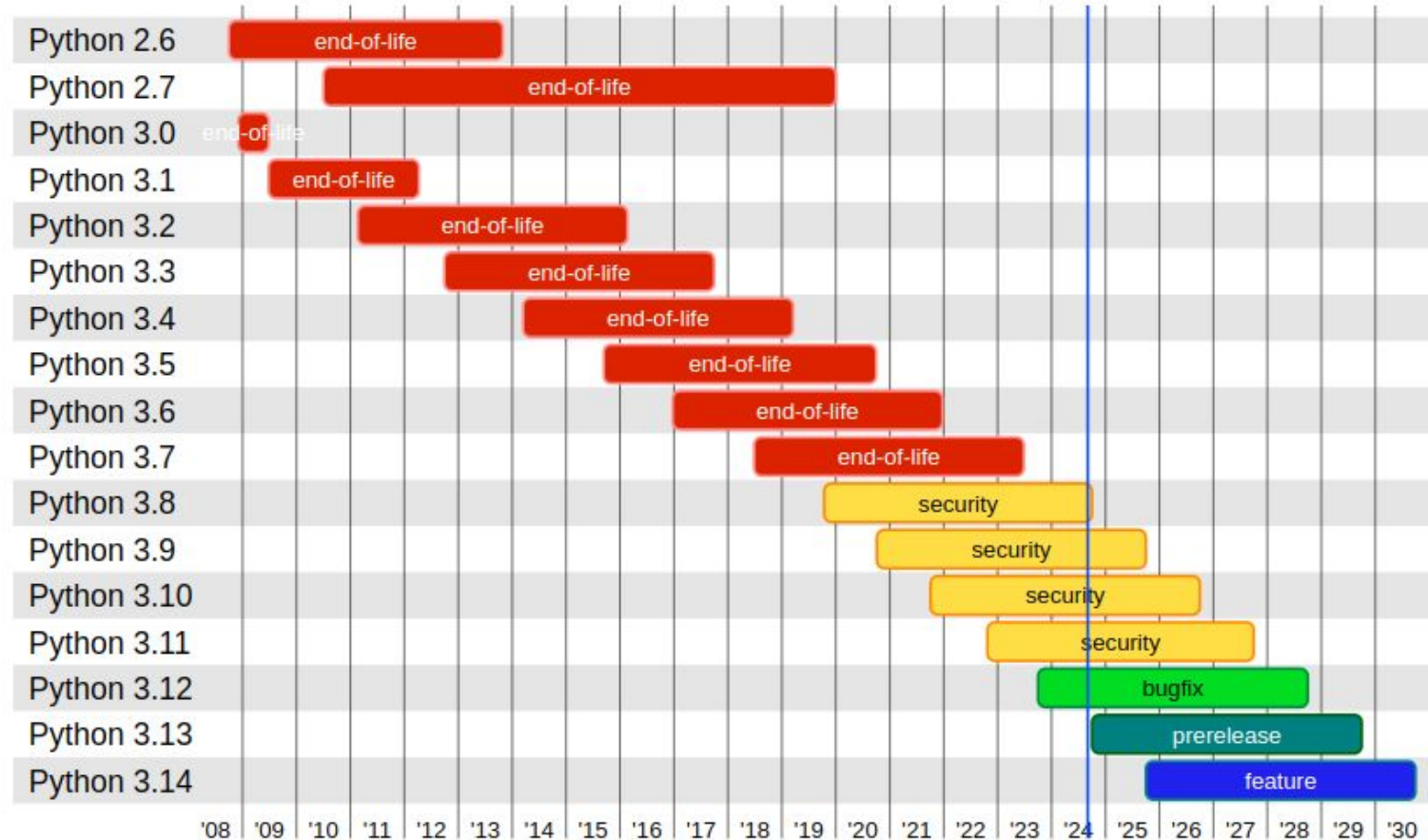


Supported versions

Dates shown in *italic* are scheduled and can be adjusted.

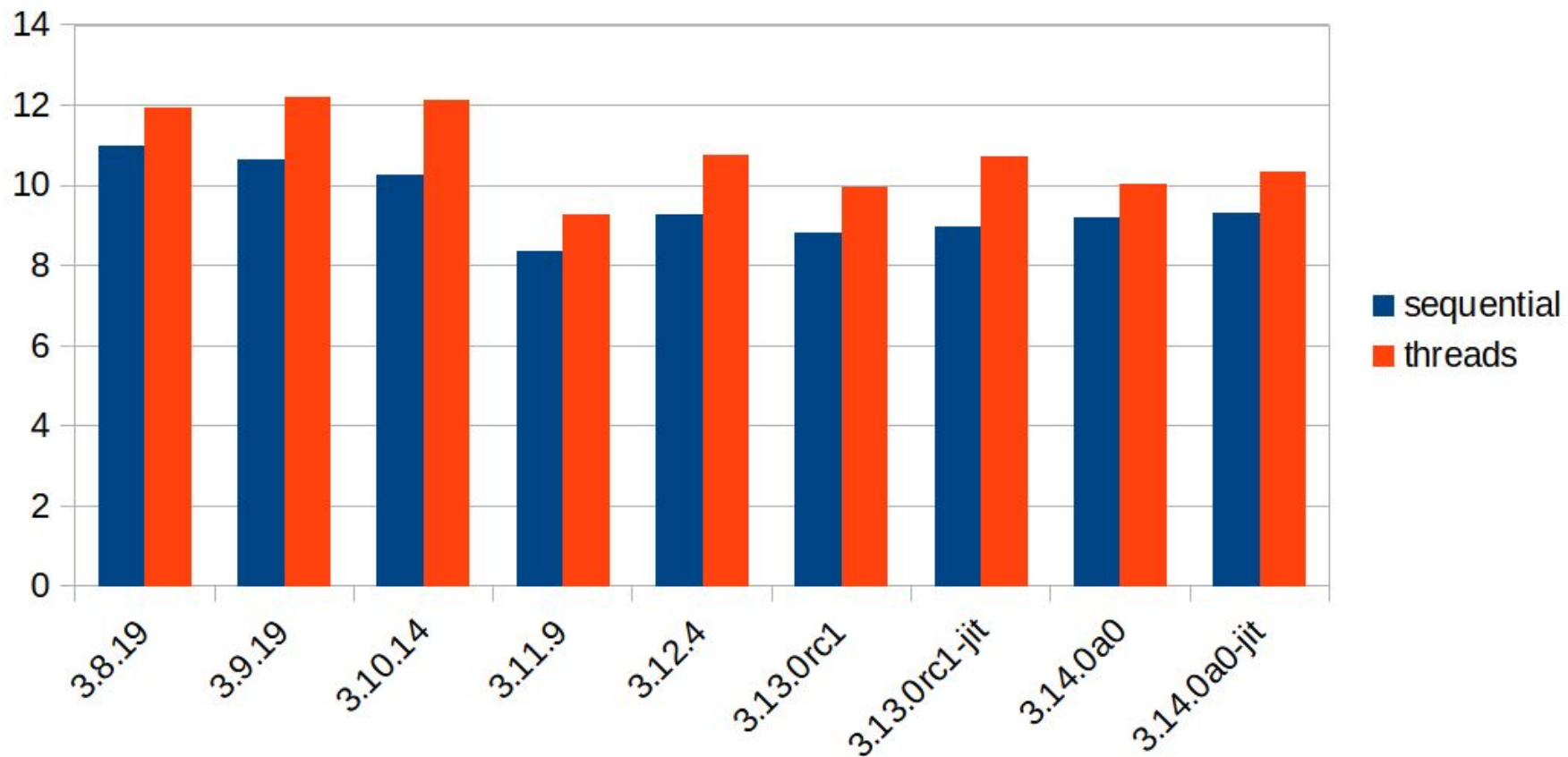
Branch	Schedule	Status	First release	End of life	Release manager
main	PEP 745	feature	<i>2025-10-01</i>	<i>2030-10</i>	Hugo van Kemenade
3.13	PEP 719	prerelease	<i>2024-10-01</i>	<i>2029-10</i>	Thomas Wouters
3.12	PEP 693	bugfix	2023-10-02	<i>2028-10</i>	Thomas Wouters
3.11	PEP 664	security	2022-10-24	<i>2027-10</i>	Pablo Galindo Salgado
3.10	PEP 619	security	2021-10-04	<i>2026-10</i>	Pablo Galindo Salgado
3.9	PEP 596	security	2020-10-05	<i>2025-10</i>	Łukasz Langa
3.8	PEP 569	security	2019-10-14	<i>2024-10</i>	Łukasz Langa

Python release cycle

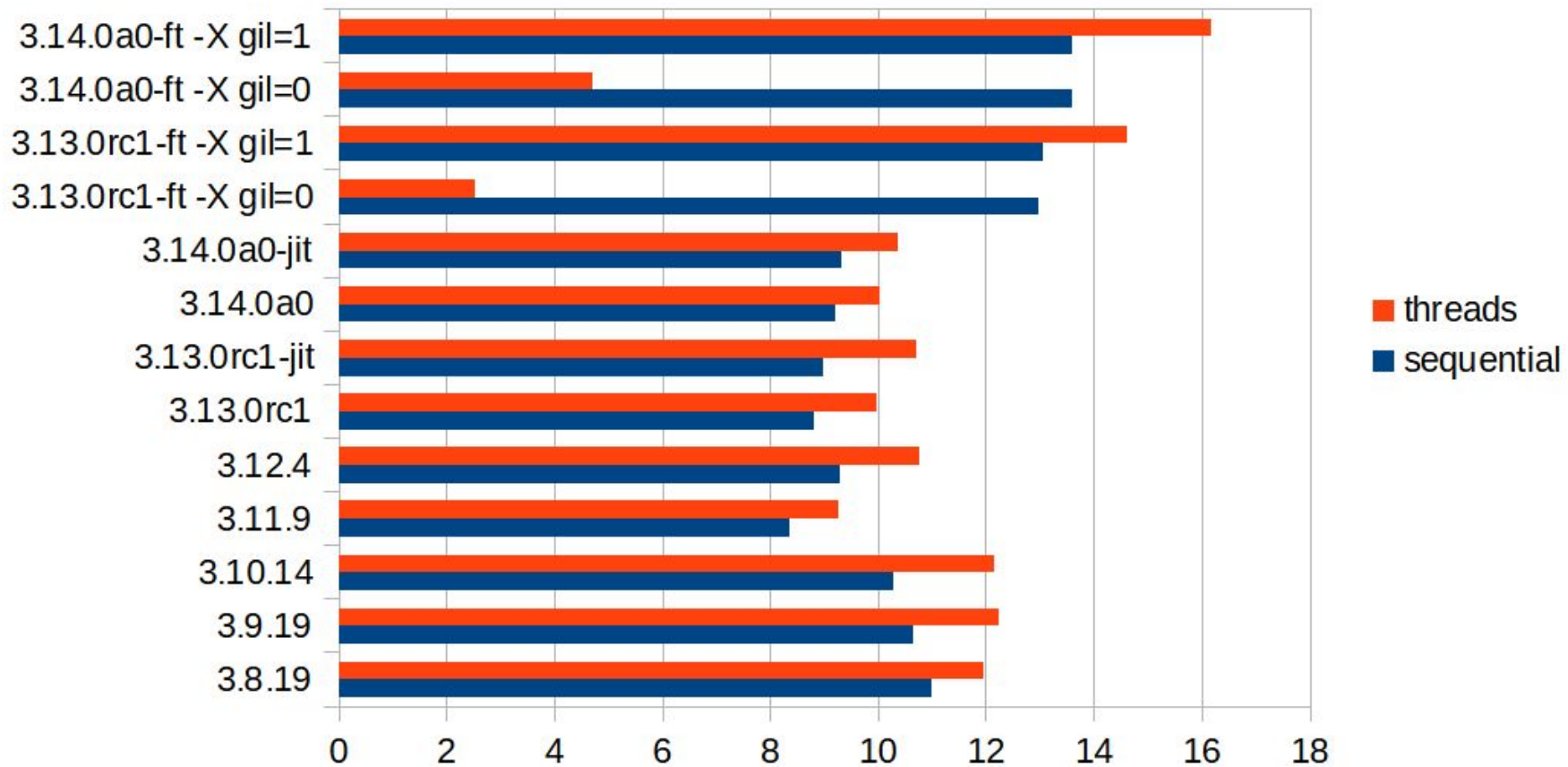


Benchmarks com Ryzen 7 (laptop)

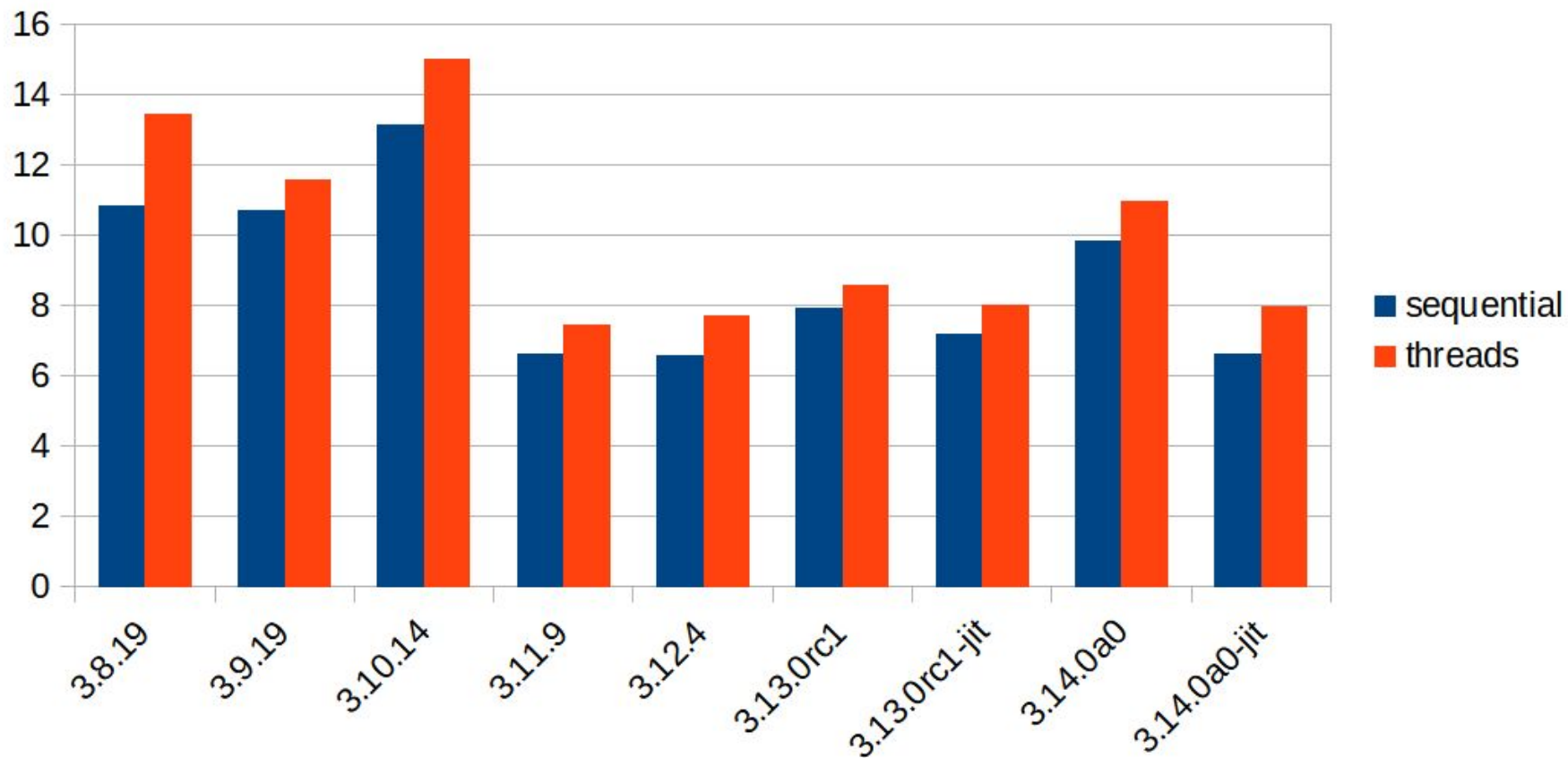
n-queens



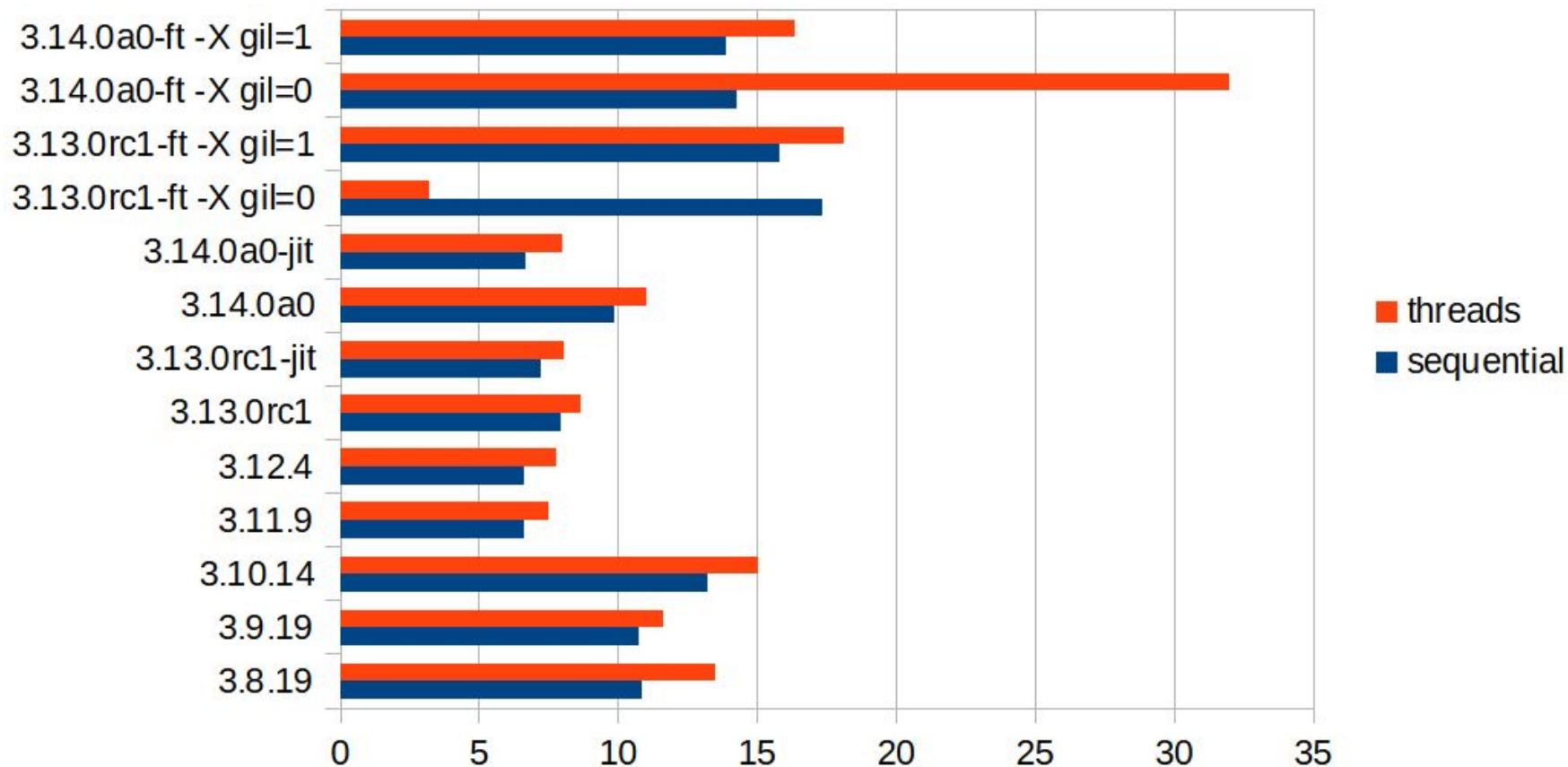
n-queens



fibonacci sequence

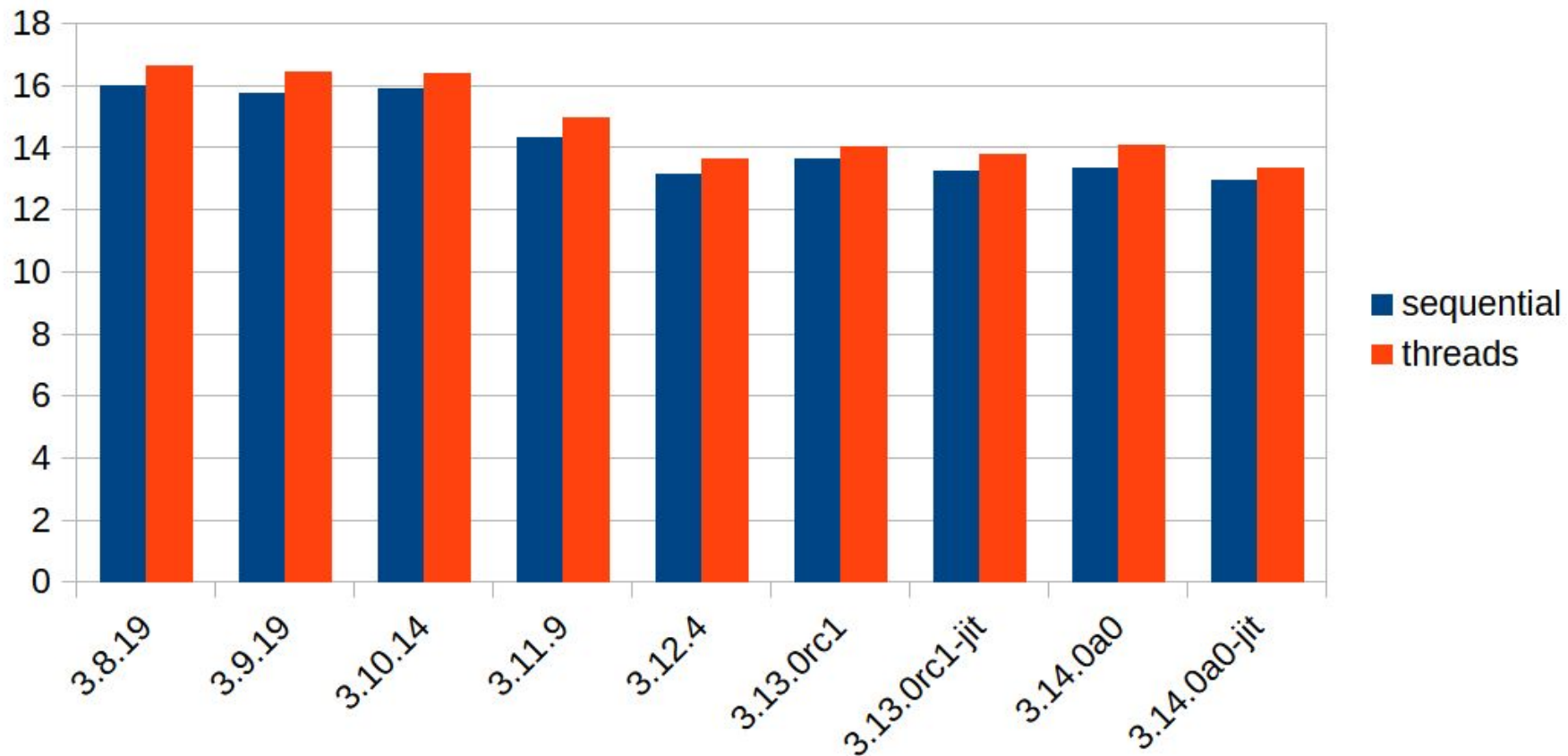


fibonacci sequence

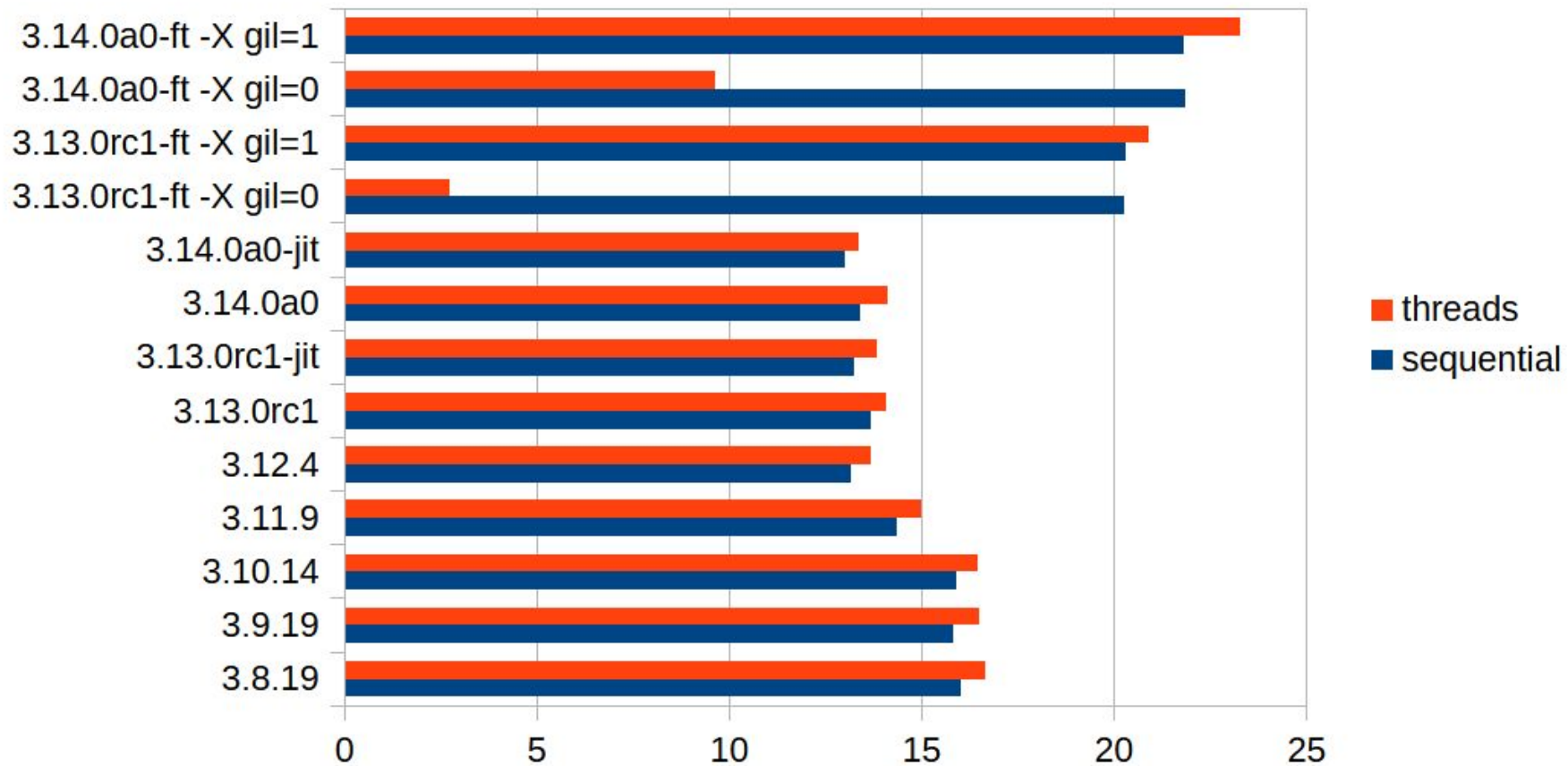


Benchmarks com Dual Xeon E5-2683 v4 (servidor)

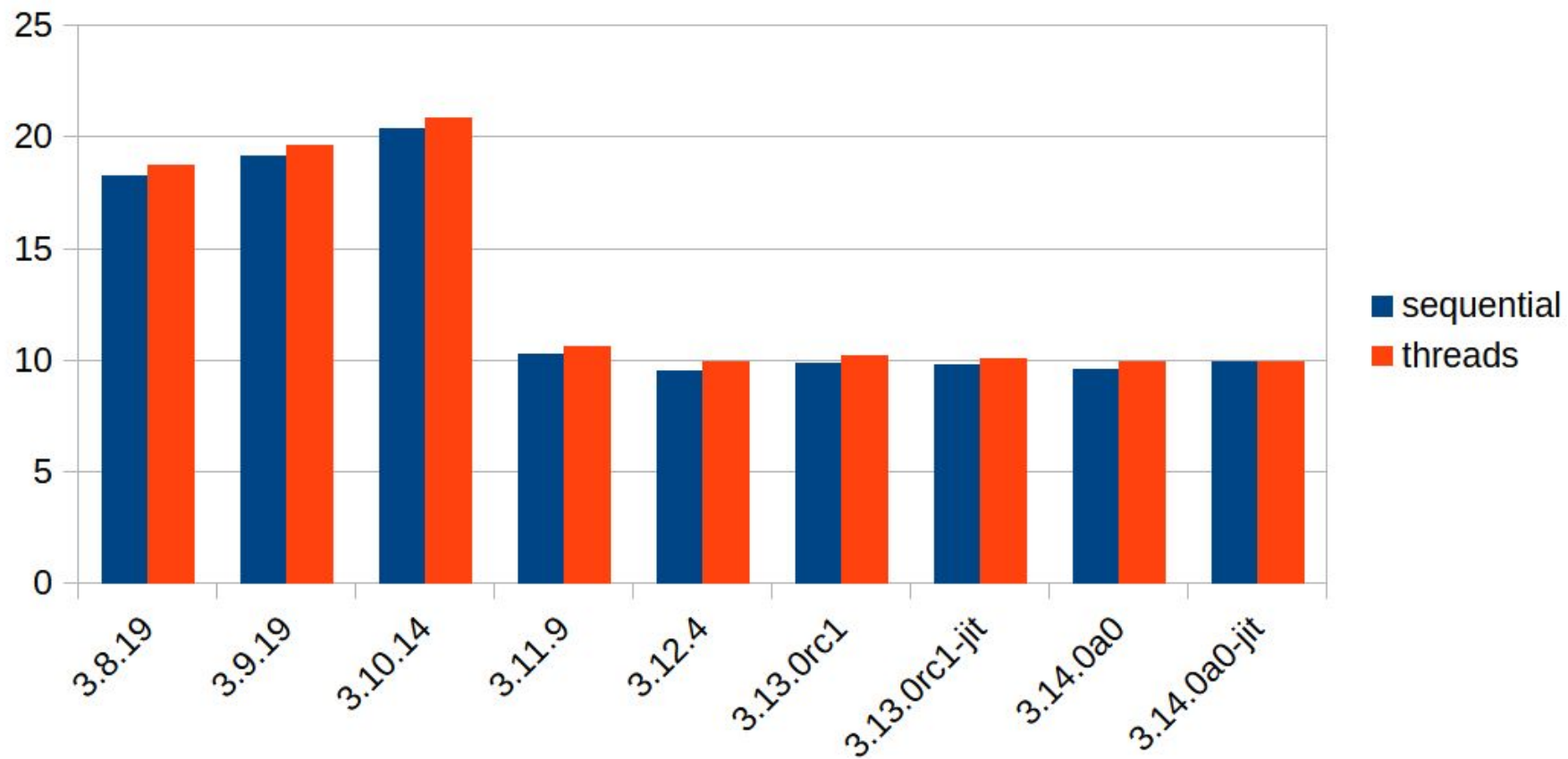
n-queens



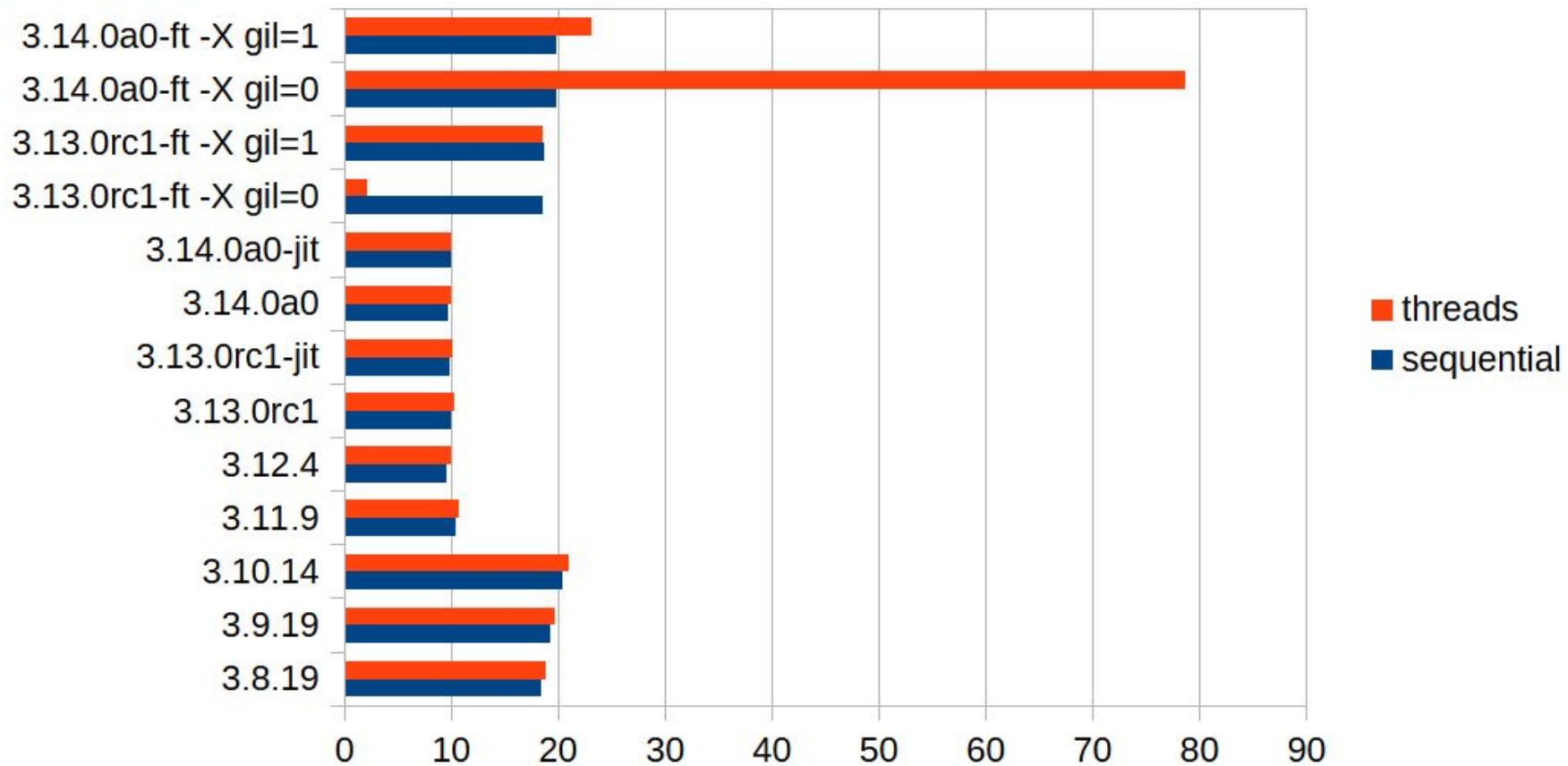
n-queens



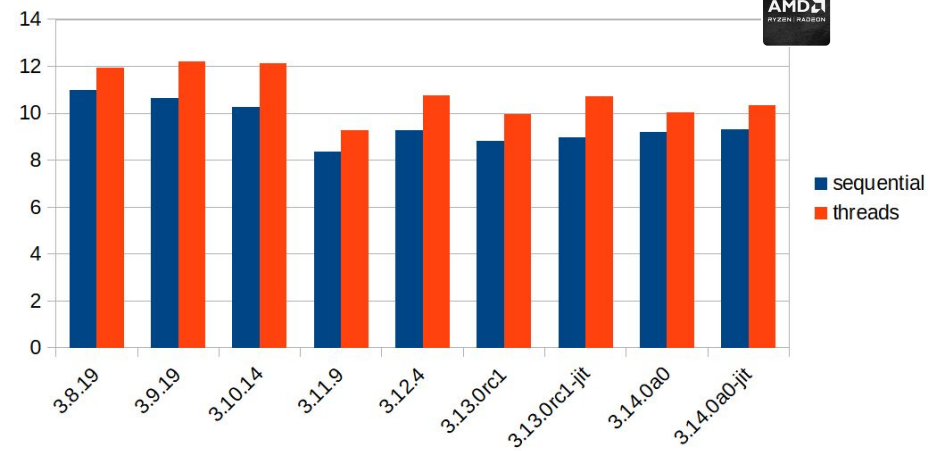
fibonacci sequence



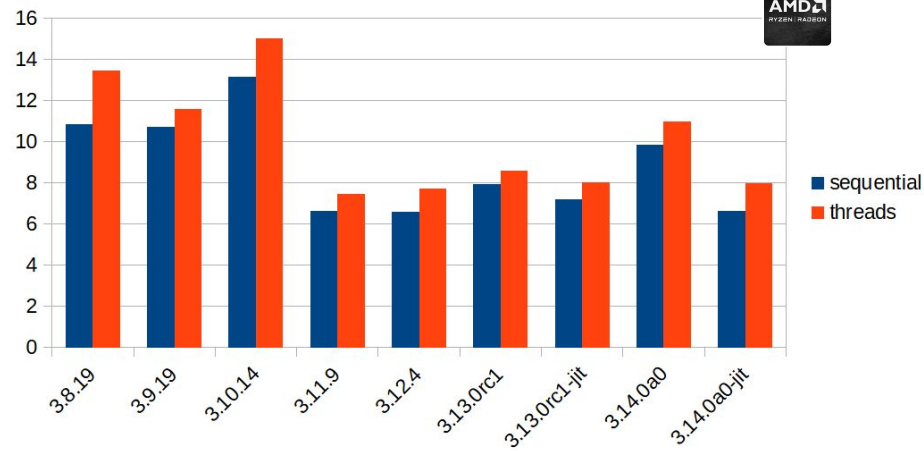
fibonacci sequence



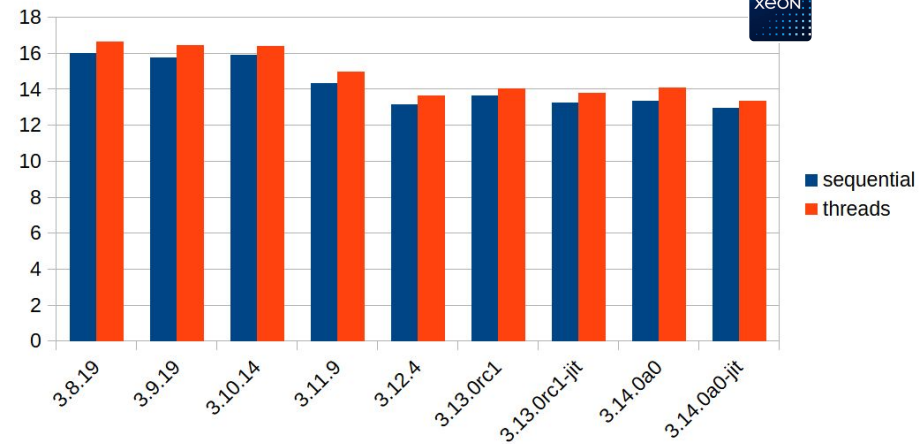
n-queens



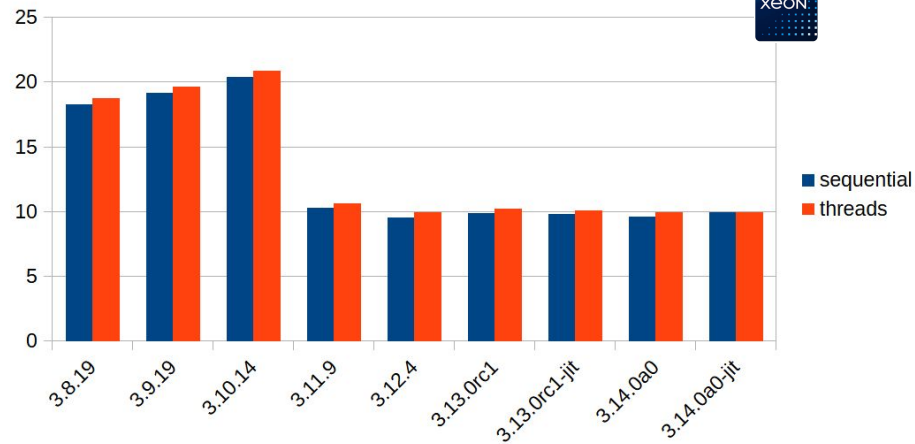
fibonacci sequence



n-queens

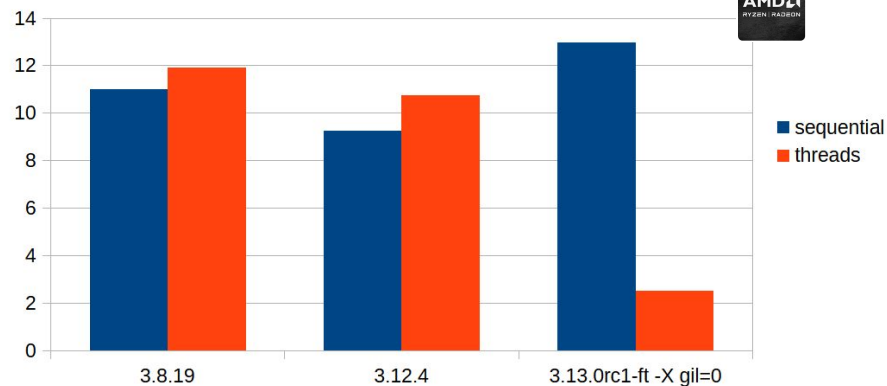


fibonacci sequence



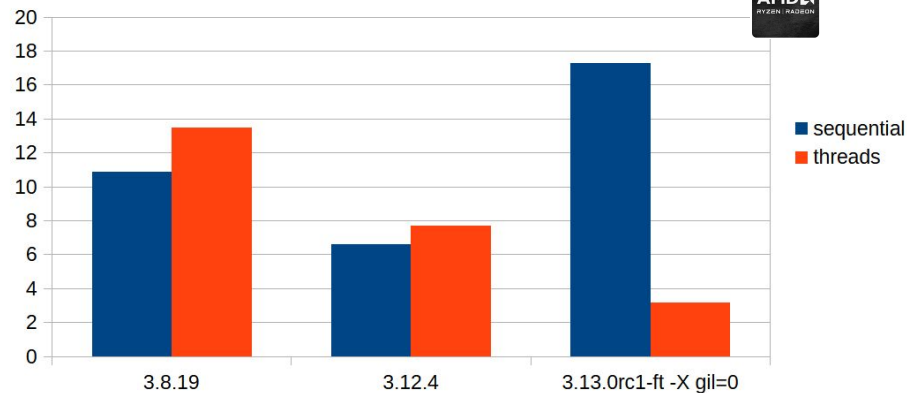
n-queens

free threading



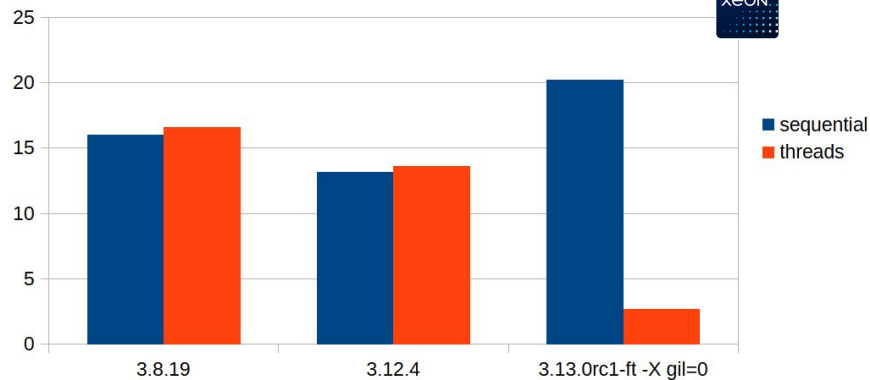
fibonacci sequence

free threading



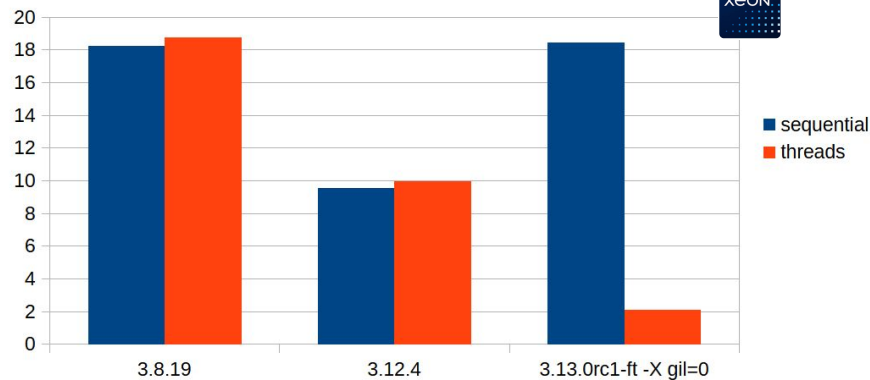
n-queens

free threading



fibonacci sequence

free threading













Obrigado

Fabiano Weimar [Xiru]

xiru@xiru.org



Sep 2024	Sep 2023	Change	Programming Language		Ratings	Change
1	1			Python	20.17%	+6.01%
2	3	▲		C++	10.75%	+0.09%
3	4	▲		Java	9.45%	-0.04%
4	2	▼		C	8.89%	-2.38%
5	5			C#	6.08%	-1.22%
6	6			JavaScript	3.92%	+0.62%
7	7			Visual Basic	2.70%	+0.48%
8	12	▲		Go	2.35%	+1.16%
9	10	▲		SQL	1.94%	+0.50%
10	11	▲		Fortran	1.78%	+0.49%