

---

---

**Programação Paralela - ELC139**

# **N-Rainhas com OpenMP**

— Rhauani Fazul e Roger Couto —

---

---

# Sumário

- O problema
- Código original
  - Primeira solução
    - Resultados
  - Segunda solução
    - Resultados
  - Distribuição
- Referências

# 0 problema

- Calcular o número de soluções distintas existentes para o posicionamento de **N** rainhas em um tabuleiro de dimensão **NxN** de tal modo que nenhuma rainha ataque a outra;
- Lógica baseada nas funções:
  - *ok()*;
  - *put\_queen()*;
  - *nqueens()*;
  - *find\_queens()*.

# Primeira solução

- Código disponível em: <https://goo.gl/XtqNZz>

- Compilação:

- Makefile:

```
$ make
```

- Diretamente:

```
$ gcc -fopenmp -o main *.c -Wall
```

- Execução:

```
$ ./main <size> <nThreads>
```

# Primeira solução

```
63 void nqueens(int size, int *solutions, int qtd_threads) {
64     int i, count = 0;
65
66     #pragma omp parallel private(i) num_threads(qtd_threads)
67     {
68         int *position = (int*) malloc(sizeof(int) * size);
69
70         #pragma omp for schedule(static, size/qtd_threads) reduction(+:count)
71         for (i = 0; i < size; i++) {
72             int j;
73             position[0] = i;
74
75             for (j = 1; j < size; j++)
76                 position[j] = -1;
77
78             int queen_number = 1;
79             while (queen_number > 0) {
80                 if ( put_queen(size, queen_number, position) ) {
81                     queen_number++;
82                     if (queen_number == size) {
83                         count += 1;
84                         position[queen_number-1] = -1;
85                         queen_number -= 2;
86                     }
87                 } else {
88                     queen_number--;
89                 }
90             }
91         }
92     }
93
94     *solutions = count;
95 }
```

# Primeira solução

- Função *nqueens()*;
- Criando o time de threads:

```
#pragma omp parallel private(i) num_threads(qtd_threads)
```

- Dividindo as iterações do laço entre o time e realizando a acumulação do número de soluções encontradas:

```
#pragma omp for schedule(static, size/qtd_threads) reduction(+:count)
```

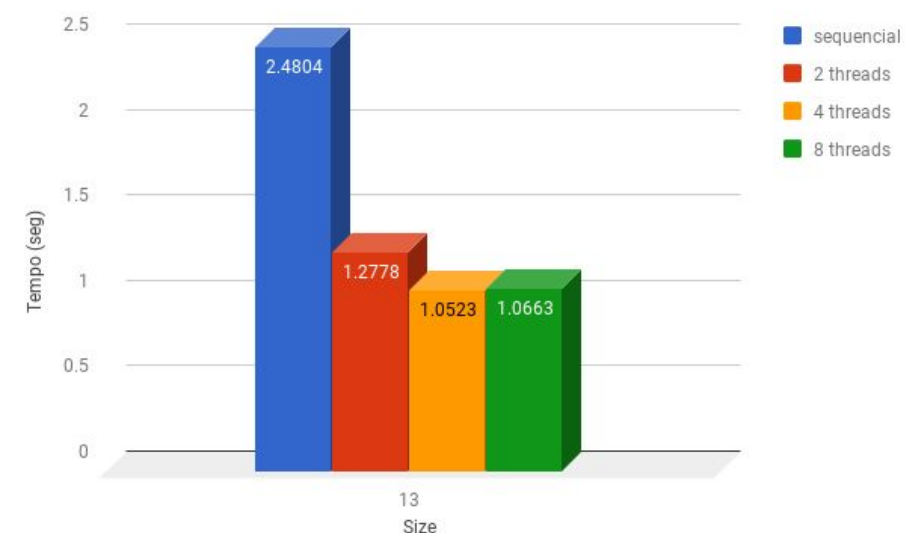
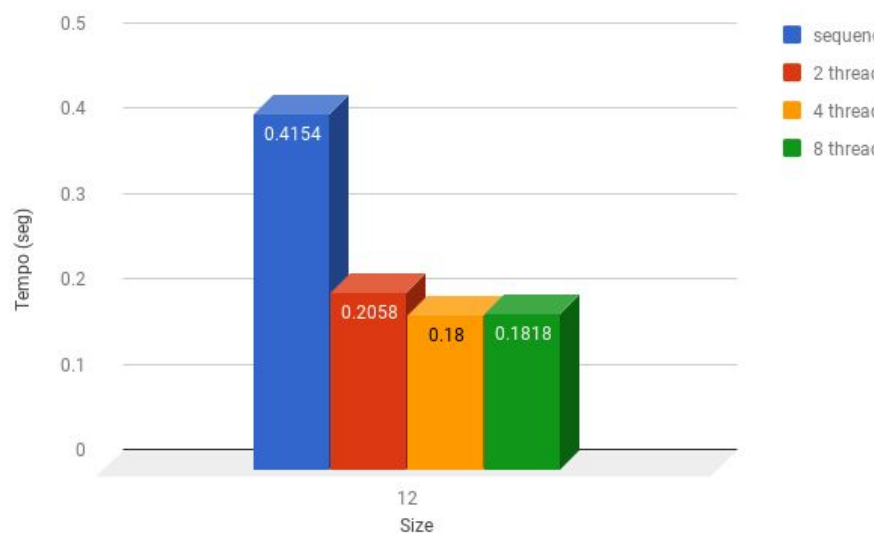
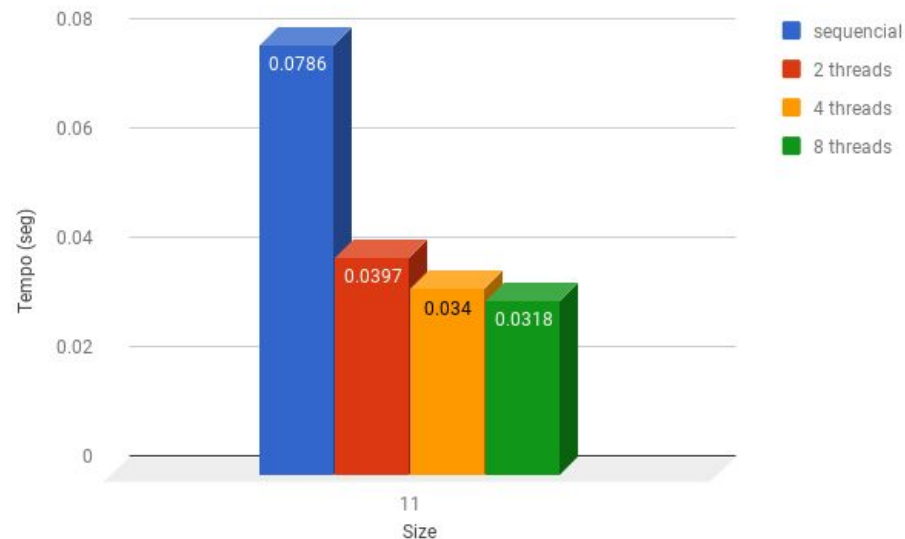
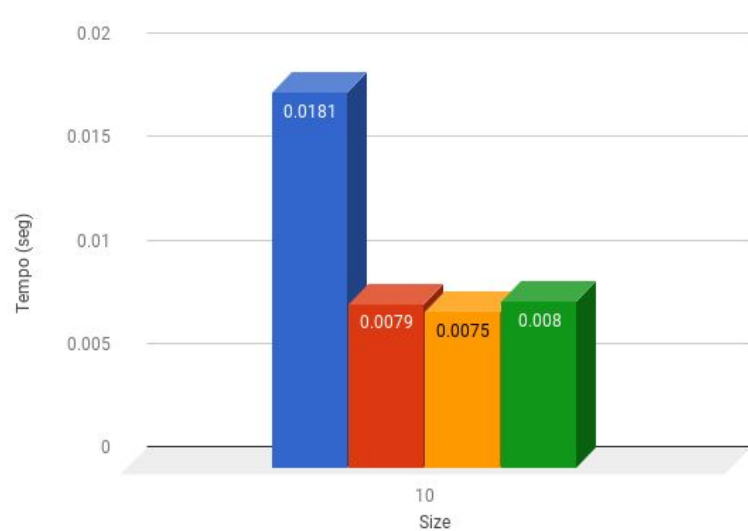
\* É necessário alocar um vetor de posições para para cada *thread*.

# Resultados

- Tempo (segundos) para obtenção da quantidade de soluções do problema das N-Rainhas.
  - Escalonamento estático.

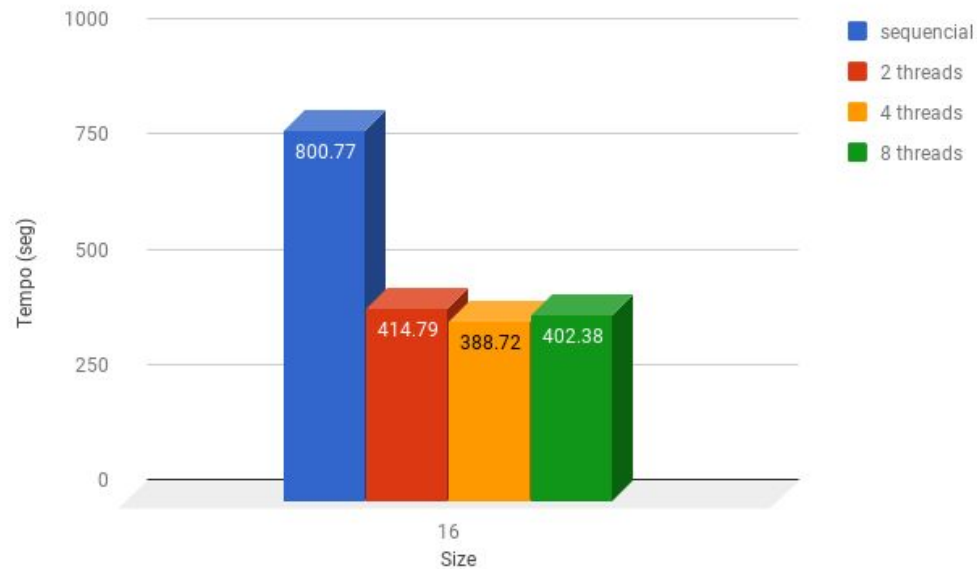
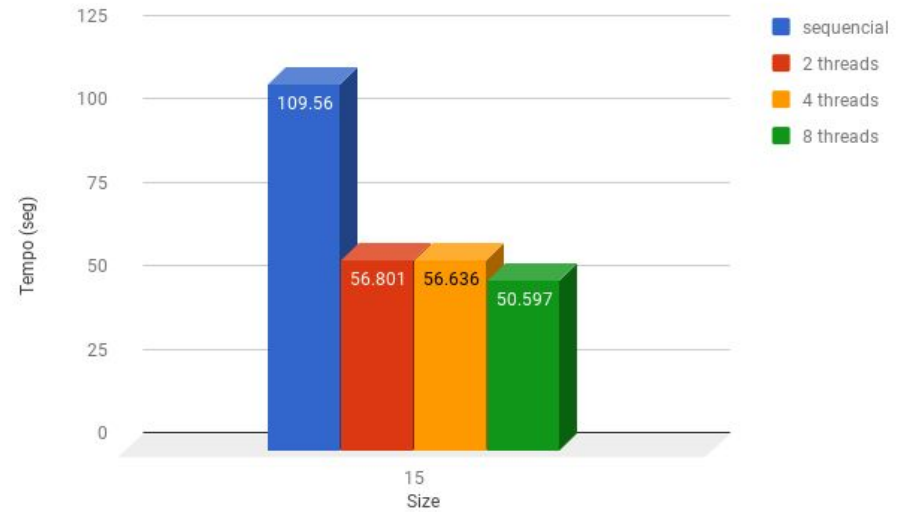
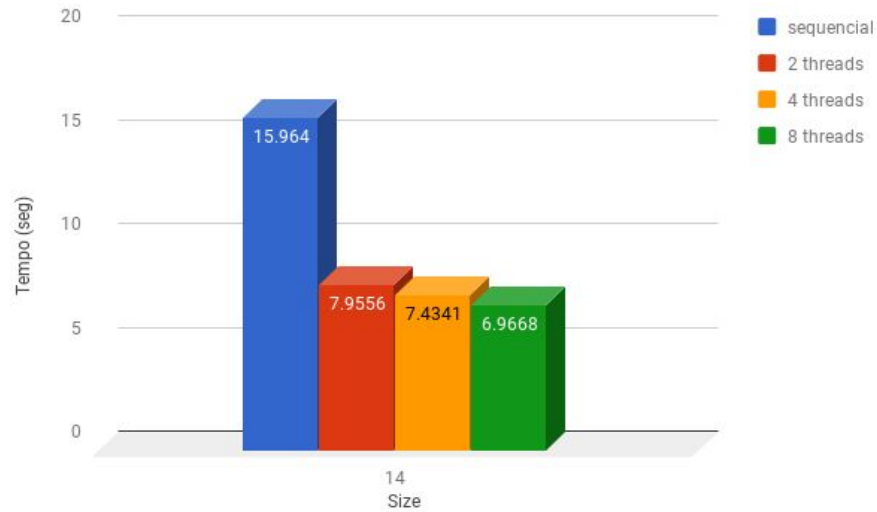
SIZE (N)	1 THREAD (SEQUENCIAL)	2 THREADS	4 THREADS	8 THREADS
10	0.0181	0.0079	0.0075	0.008
11	0.0786	0.0397	0.034	0.0318
12	0.4154	0.2058	0.18	0.1818
13	2.4804	1.2778	1.0523	1.0663
14	15.964	7.9556	7.4341	6.9668
15	109.56	56.801	56.636	50.597
16	800.77	414.79	388.72	402.38

# Resultados





# Resultados



# Resultados

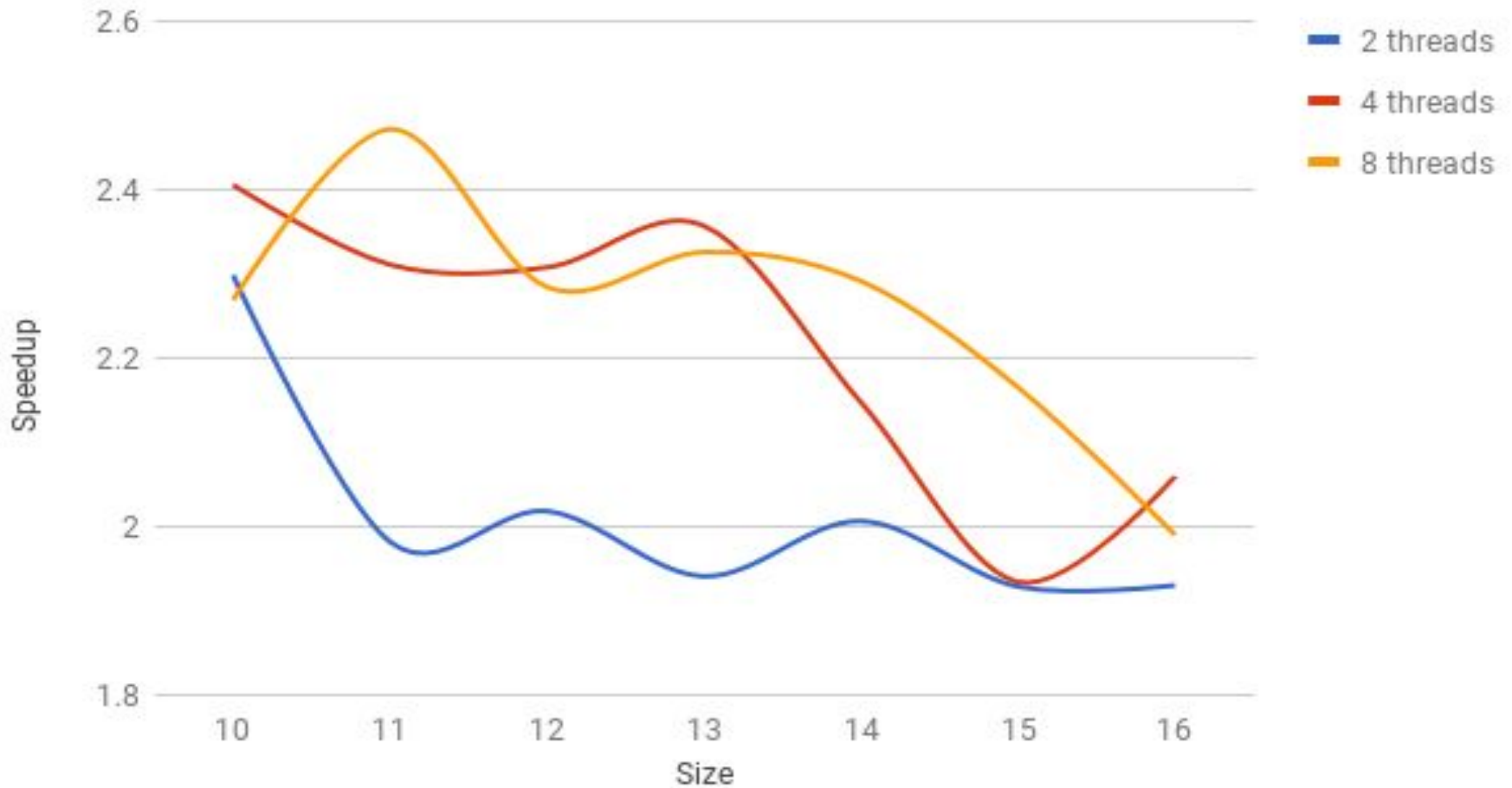
- *Speedup* para obtenção da quantidade de soluções do problema das N-Rainhas.
  - Escalonamento estático.

SIZE (N)	2 THREADS	4 THREADS	8 THREADS
10	2.2986	2.4059	2.2691
11	1.9819	2.3111	2.4722
12	2.0185	2.308	2.2845
13	1.9412	2.3571	2.3261
14	2.0067	2.1475	2.2915
15	1.9289	1.9345	2.1654
16	1.9305	2.06	1.9901

# Resultados

- Escalonamento estático.

Desempenho OpenMP com schedule estático



# Segunda solução

- Código disponível em: <https://goo.gl/XtqNZz>
  - **nqueens.c**: descomentar [linha 70](#) e comentar [linha 71](#).

- Compilação:

- Makefile:

```
$ make
```

- Diretamente:

```
$ gcc -fopenmp -o main *.c -Wall
```

- Execução:

```
$ ./main <size> <nThreads>
```

# Segunda solução

- Função *nqueens()*;
- Criando o time de threads:

```
#pragma omp parallel private(i) num_threads(qtd_threads)
```

- Dividindo as iterações do laço entre o time e realizando a acumulação do número de soluções encontradas:

```
#pragma omp for schedule(dynamic) reduction(+:count)
```

\* É necessário alocar um vetor de posições para para cada *thread*.

# Resultados

- Tempo (segundos) para obtenção da quantidade de soluções do problema das N-Rainhas.
  - Escalonamento dinâmico.

SIZE (N)	1 THREAD (SEQUENCIAL)	2 THREADS	4 THREADS	8 THREADS
10	0.0181	0.0085	0.0073	0.0067
11	0.0786	0.0408	0.0329	0.0323
12	0.4154	0.2077	0.1756	0.1747
13	2.4804	1.3117	1.0949	1.0827
14	15.964	8.076	7.365	7.691
15	109.56	63.45	56.49	55.73
16	800.77	460.44	395.08	396.46

# Resultados

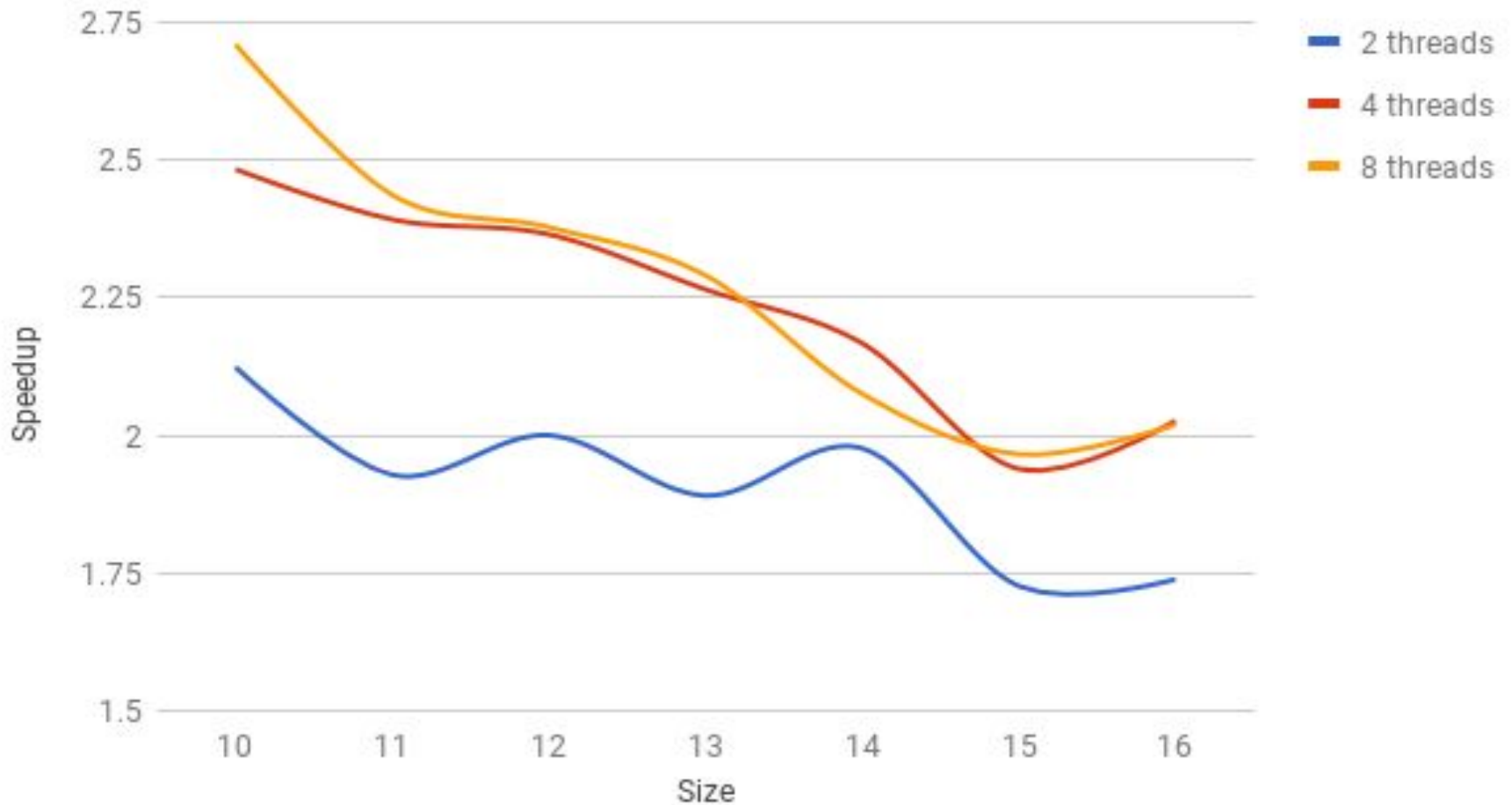
- *Speedup* para obtenção da quantidade de soluções do problema das N-Rainhas.
  - Escalonamento dinâmico.

SIZE (N)	2 THREADS	4 THREADS	8 THREADS
10	2.1243	2.4836	2.7104
11	1.9287	2.3924	2.4369
12	2.0006	2.3653	2.3782
13	1.891	2.2654	2.291
14	1.9768	2.1676	2.0758
15	1.7267	1.9394	1.966
16	1.7391	2.0269	2.0198

# Resultados

- Escalonamento dinâmico.

Desempenho OpenMP com schedule dinâmico



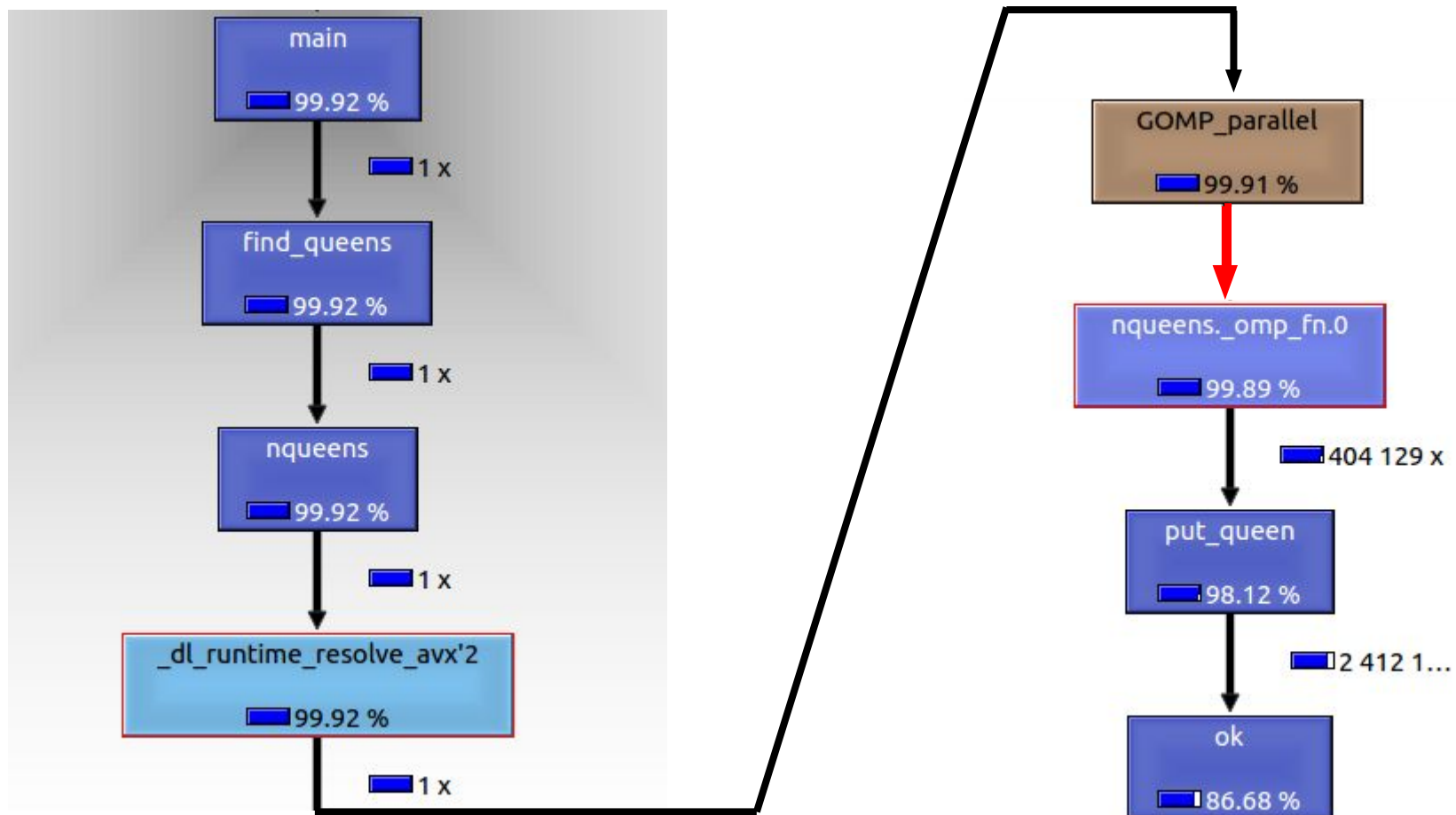


# Distribuição

- Também foi realizada uma breve análise sobre a ocupação e distribuição de trabalho de cada *thread* nas diferentes funções;
- Para exemplificar utilizou-se 4 *threads* e **N** = 14.

# Distribuição

- Lógica geral do programa paralelizado com OpenMP:











# Distribuição







## nqueens.\_omp\_fn.0 <cycle 1>

Profile Part	Incl.	Self	Called	Comment
PID 15716, section 1, thread 1	 99.90	1.77	1	Program termination
PID 15716, section 1, thread 2	 99.98	1.70	1	Program termination
PID 15716, section 1, thread 3	 99.93	1.70	1	Program termination
PID 15716, section 1, thread 4	 100.00	1.77	1	Program termination

## put\_queen

Profile Part	Incl.	Self	Called	Comment
PID 15716, section 1, thread 1	 98.13	 11.44	404 165	Program termination
PID 15716, section 1, thread 2	 98.28	 10.93	437 817	Program termination
PID 15716, section 1, thread 3	 98.23	 10.92	437 817	Program termination
PID 15716, section 1, thread 4	 98.23	 11.42	404 165	Program termination

## ok

Profile Part	Incl.	Self	Called	Comment
PID 15716, section 1, thread 1	 86.69	 86.69	2 412 373	Program termination
PID 15716, section 1, thread 2	 87.35	 87.35	2 602 639	Program termination
PID 15716, section 1, thread 3	 87.31	 87.31	2 600 282	Program termination
PID 15716, section 1, thread 4	 86.81	 86.81	2 410 462	Program termination

- Incl: custo incluindo todas as funções chamadas ('Inclusive Cost');
- Self: somente o custo da função em si ('Self Cost').

# Distribuição

## nqueens.\_omp\_fn.0 <cycle 1>

Profile Part	Incl.	Self	Called	Comment
PID 7298, section 1, thread 1	■ 99.94	1.72	1	Program termination
PID 7298, section 1, thread 2	■ 99.99	1.72	1	Program termination
PID 7298, section 1, thread 3	■ 99.96	1.78	1	Program termination
PID 7298, section 1, thread 4	■ 99.95	1.74	1	Program termination

## put\_queen

Profile Part	Incl.	Self	Called	Comment
PID 7298, section 1, thread 1	■ 98.22	■ 11.07	558 972	Program termination
PID 7298, section 1, thread 2	■ 98.27	■ 11.05	574 520	Program termination
PID 7298, section 1, thread 3	■ 98.18	■ 11.55	257 858	Program termination
PID 7298, section 1, thread 4	■ 98.21	■ 11.24	292 614	Program termination

## ok

Profile Part	Incl.	Self	Called	Comment
PID 7298, section 1, thread 1	■ 87.15	■ 87.15	3 326 748	Program termination
PID 7298, section 1, thread 2	■ 87.21	■ 87.21	3 416 424	Program termination
PID 7298, section 1, thread 3	■ 86.63	■ 86.63	1 539 703	Program termination
PID 7298, section 1, thread 4	■ 86.97	■ 86.97	1 742 881	Program termination

- Incl: custo incluindo todas as funções chamadas ('Inclusive Cost');
- Self: somente o custo da função em si ('Self Cost').

# Terceira solução

- Código disponível em: <https://goo.gl/vW15SQ>

- Compilação:

- Makefile:

```
$ make
```

- Diretamente:

```
$ gcc -fopenmp -o main *.c -Wall
```

- Execução:

```
$ ./main <size> <nThreads>
```

# Terceira solução

- Esta solução utiliza a seguinte abordagem:
  - cria-se uma thread para cada posição possível da primeira rainha na primeira coluna;
  - a partir desta rainha verifica-se (em cada thread) quantas soluções possíveis para cada variação da primeira rainha existem.

# Terceira solução

```
void nqueens(int size, int *solutions) {
    int count;
    int* position;
    count = 0;
    //Cria uma thread para cada posição possível da primeira rainha
    #pragma omp parallel num_threads(size) shared(solutions) private(position)
    {
        int j;
        position = (int *) malloc(size * sizeof(int));
        position[0] = omp_get_thread_num(); //Adiciona a posição da primeira rainha conforme o número da thread
        for(j = 1; j < size; j++)
            position[j] = -1;
        int queen_number = 1;
        while(queen_number > 0) {
            if(put_queen(size, queen_number, position)) {
                queen_number++;
                if(queen_number == size) {
                    #pragma omp critical
                    {
                        count += 1;
                        position[queen_number-1] = -1;
                        queen_number -= 2;
                    }
                }
            }
            else {
                queen_number--;
            }
        }
    }
    *solutions = count;
}
```

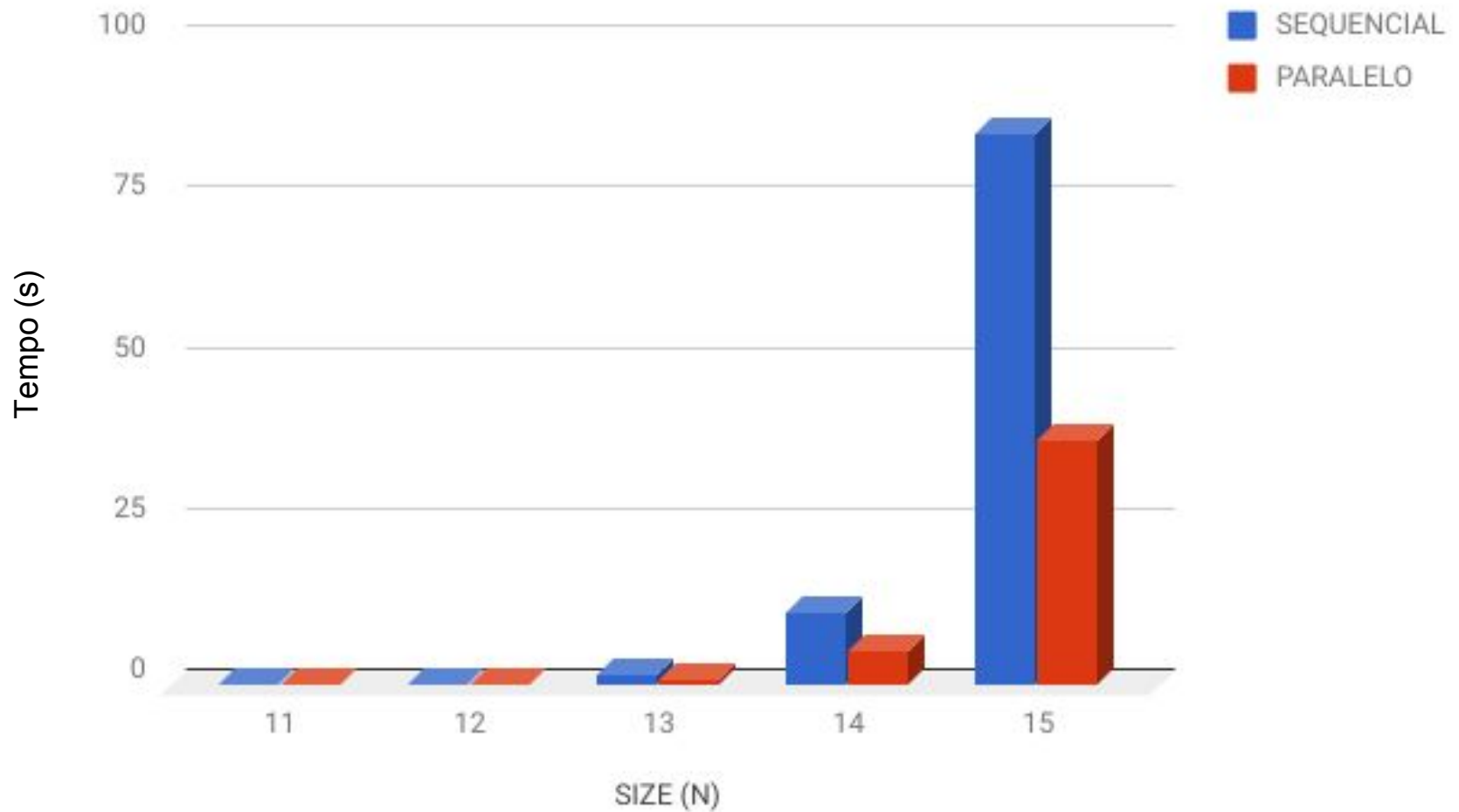
# Resultados

- Tempo (segundos) e *Speedup* para obtenção da quantidade de soluções do problema das N-Rainhas.

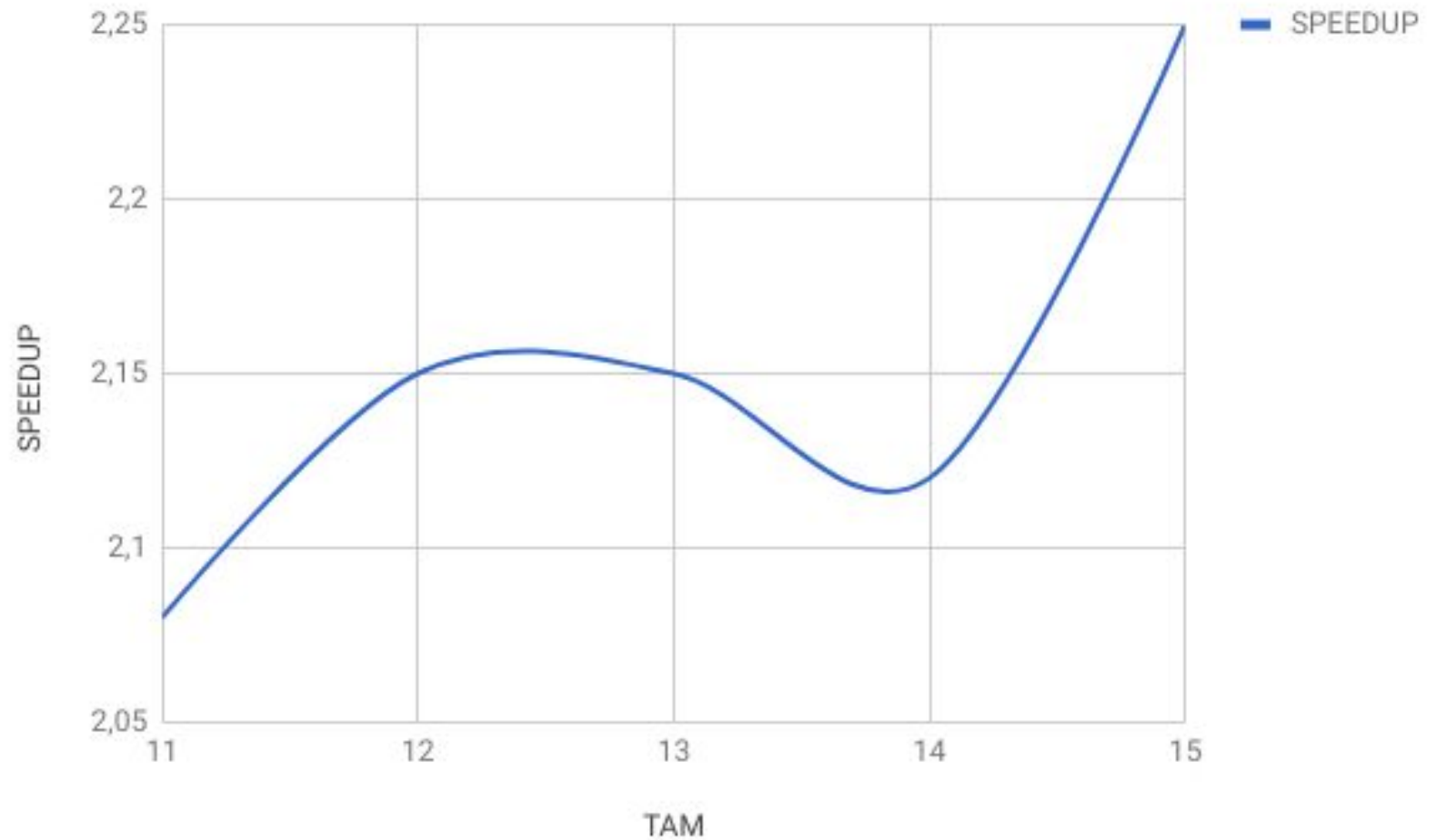
SIZE (N)	SEQUENCIAL	PARALELO	<i>SPEEDUP</i>
11	0,0584	0,0280	2,08
12	0,2942	0,1369	2,14
13	1,7335	0,8073	2,14
14	11,405	5,3693	2,12
15	85,605	38,025	2,25



# Resultados



# Resultados



# Quarta solução

- Código disponível em: <https://goo.gl/Ufb5ky>

- Compilação:

- Makefile:

```
$ make
```

- Diretamente:

```
$ gcc -fopenmp -o main *.c -Wall
```

- Execução:

```
$ ./main <size> <nThreads>
```

# Quarta solução

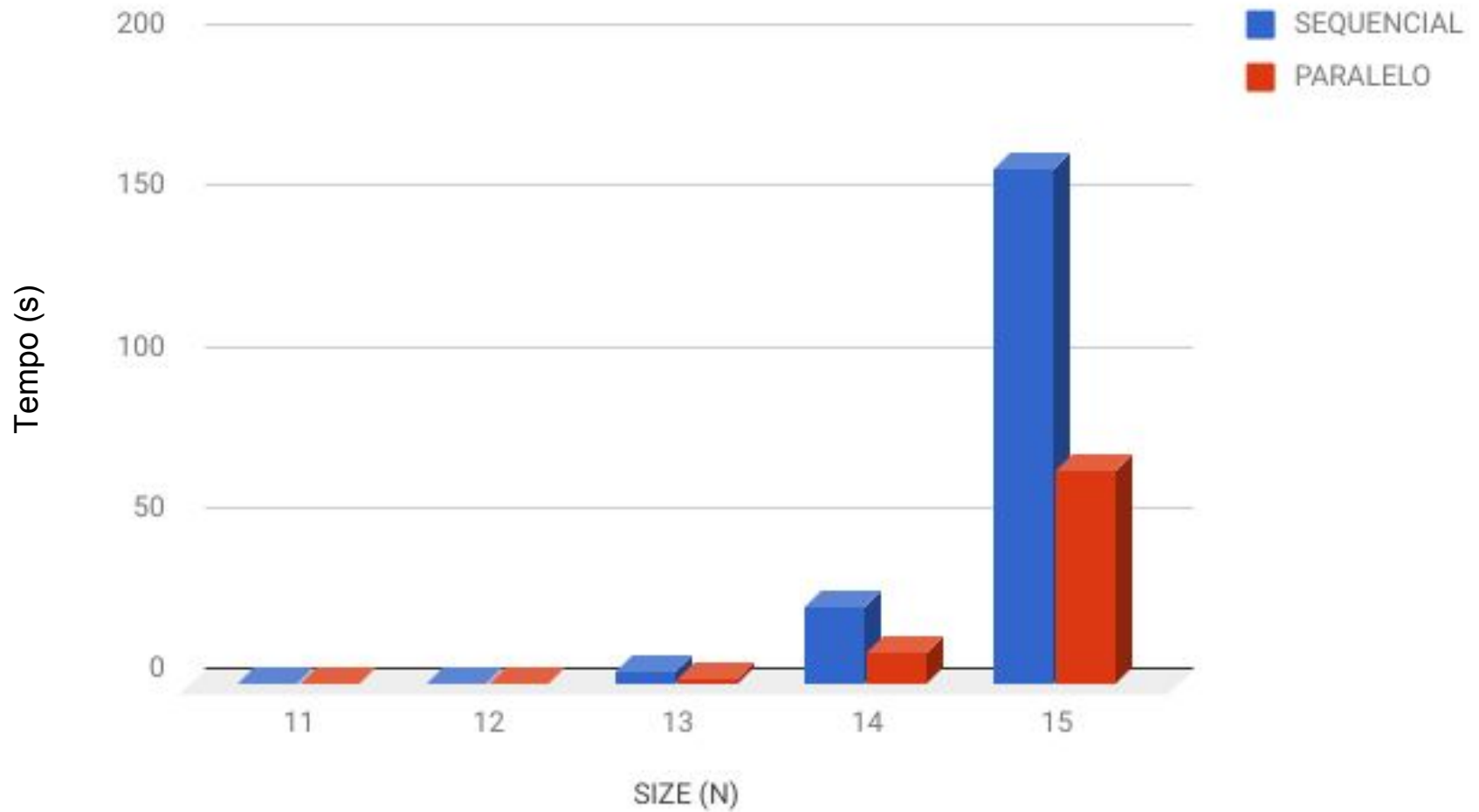
- A quarta solução foi um código novo, que faz uso de uma matriz para guardar as posições e uma pilha pra voltar a partir da última posição marcada;
- Porém, o desempenho desta solução é inferior ao das soluções anteriores.

# Quarta solução

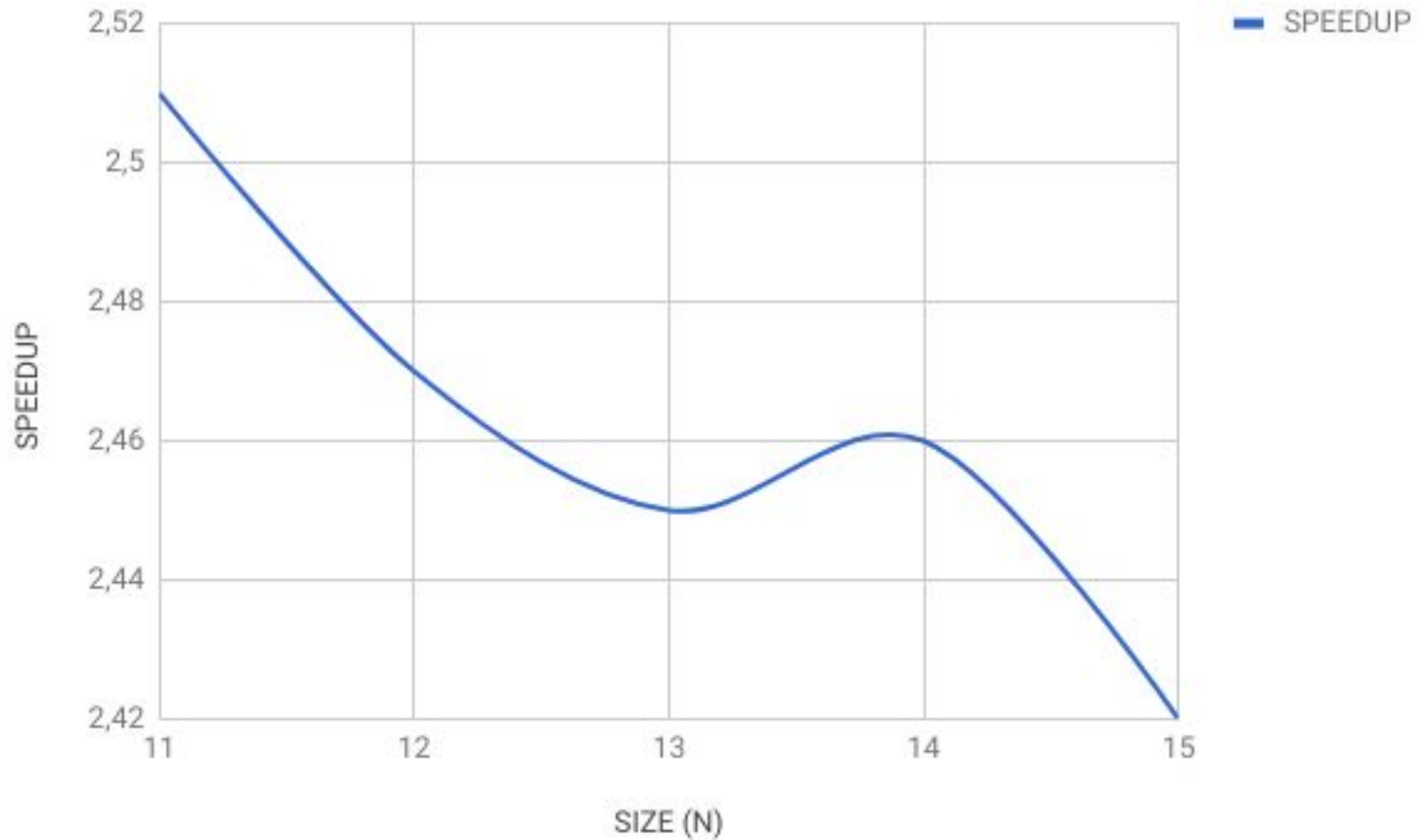
- Tempo (segundos) e *Speedup* para obtenção da quantidade de soluções do problema das N-Rainhas.

SIZE (N)	SEQUENCIAL	PARALELO	<i>SPEEDUP</i>
11	0,1207	0,0482	2,51
12	0,6352	0,2574	2,47
13	3,8738	1,5788	2,45
14	24,1534	9,807	2,46
15	160,1525	66,1827	2,42

# Resultados



# Resultados



# Referências

- FSU Department of Science Computing. **C++ Examples of Parallel Programming with OpenMP.** <https://goo.gl/sqmujr>
- Google Optimization Tools. **The N-queens Problem.** <https://goo.gl/fqS6sW>
- Lawrence Livermore National Laboratory. **OpenMP.** <https://goo.gl/o2wTxR>
- Mark Bull. **OpenMP Tips, Tricks and Gotchas.** <https://goo.gl/L9Xhyp>
- OpenMP. **OpenMP C and C++ Application Program Interface.** <https://goo.gl/wPbQCn>
- OpenMP. **Summary of OpenMP 3.0 C/C++ Syntax.** <https://goo.gl/VdvSpi>
- Texas Advanced Computing Center. **OpenMP topic: Reductions.** <https://goo.gl/BbMrFP>
- Wikipedia. **Eight queens puzzle.** <https://goo.gl/wnDqp2>



# Obrigado!



Perguntas?