

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and edges. Some nodes are highlighted with blue circles, and others with blue dots. The diagram is rendered in a light gray color.

N-Rainhas com OpenMP


Lucas Ferreira da Silva

A decorative network diagram in the bottom-right corner, featuring a complex web of interconnected nodes and edges. Some nodes are highlighted with blue circles, and others with blue dots. The diagram is rendered in a light gray color.

Carga de trabalho

- ◎ Concentrada na função ***nqueens***;
- ◎ Múltiplas **chamadas a funções secundárias** dentro da função *nqueens*;
- ◎ Variável ***count***;
- ◎ **Desbalanço de carga** entre os dois **laços aninhados**, sendo o **while** o mais custoso e de **perfil de execução variável**.

```
void nqueens(int size, int *solutions) {  
    int i, count;  
    int* position;  
  
    count = 0;  
  
    for(i=0; i<size; i++) {  
        int j;  
        position = (int *) malloc(size * sizeof(int));  
        position[0] = i;  
  
        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) {  
                    count += 1;  
                    position[queen_number-1] = -1;  
                    queen_number -= 2;  
                }  
            } else {  
                queen_number--;  
            }  
        }  
    }  
  
    *solutions = count;  
}
```

A decorative background featuring a network diagram with nodes and connecting lines, primarily located in the top-left and bottom-right corners.

1ª Implementação com OpenMP

Implementação 1

- ◎ Paralelização do **laço mais externo**;
- ◎ **Declaração** e inicialização do **array *position* dentro do laço**;
- ◎ Definição do *schedule* ***dynamic***;
- ◎ Tratamento da **exclusão mútua do count** com a **cláusula *atomic***.

```
void nqueens(int size, int *solutions) {
    int i, count;

    count = 0;

    #pragma omp parallel for private(i) schedule(dynamic)
    for(i=0; i<size; i++) {
        int j;
        int* position = (int *) malloc(size * sizeof(int));
        position[0] = i;


        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 atomic
                    count += 1;

                    position[queen_number-1] = -1;
                    queen_number -= 2;
                }
            } else {
                queen_number--;
            }
        }
    }

    *solutions = count;
}
```

A decorative background featuring a network diagram with nodes and connecting lines, primarily located in the top-left and bottom-right corners.

2ª Implementação com OpenMP

Implementação 2

- ◎ Paralelização do **laço mais externo**;
- ◎ **Declaração** e inicialização do **array *position* dentro do laço**;
- ◎ Definição do *schedule* ***static***;
- ◎ Tratamento da **exclusão mútua do count** com a **cláusula *atomic***.


```
void nqueens(int size, int *solutions) {
    int i, count;

    count = 0;

    #pragma omp parallel for private(i) schedule(static)
    for(i=0; i<size; i++) {
        int j;
        int* position = (int *) malloc(size * sizeof(int));
        position[0] = i;

        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 atomic
                    count += 1;

                    position[queen_number-1] = -1;
                    queen_number -= 2;
                }
            } else {
                queen_number--;
            }
        }
    }

    *solutions = count;
}
```

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are larger and have concentric circles inside, suggesting a hierarchical or multi-layered structure. The lines connecting the nodes are thin and grey, creating a mesh-like pattern.

Experimentos

Experimentos

- ◎ 10 execuções de cada configuração;
- ◎ Execução das 3 versões:
 - **Serial**
 - **nqueens_OMP1**
 - **nqueens_OMP2**
- ◎ Variação do parâmetro **N** de **12 a 16**

Experimentos

◎ 3 variações do número de threads:

- 2 threads;
- 4 threads;
- 6 threads;

◎ Hardware:

- Intel® Core™ i5-2410M
- 2.30GHz
- 2 Cores
- 4 Threads
- 6 GB de RAM

◎ Sistema Operacional:

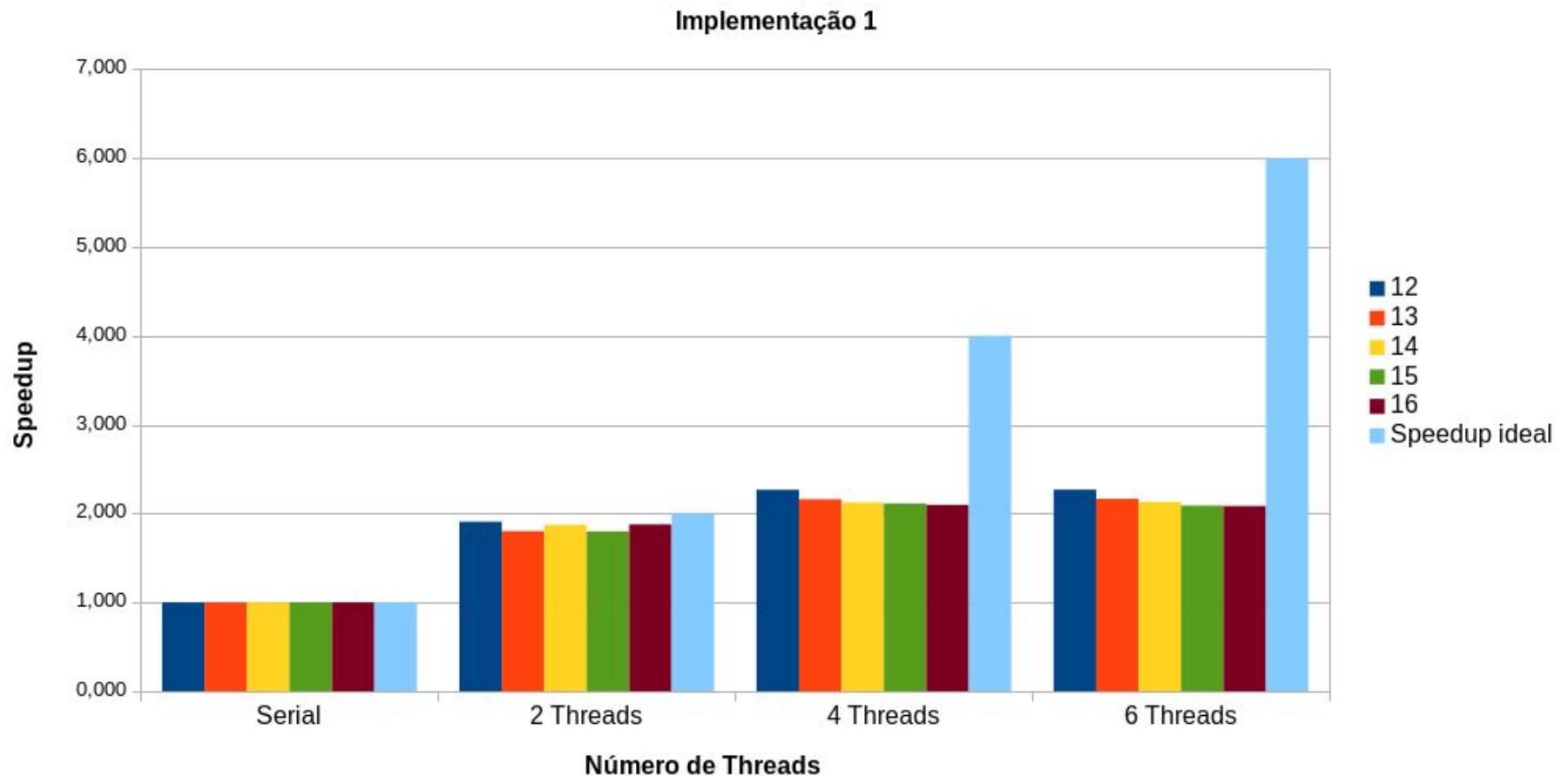
- Debian GNU/Linux Buster
- Versão do Linux: 4.15.0-2-amd64
- Versão do gcc: 7.3.0

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by small circles, some of which are larger and have concentric circles, suggesting different levels of connectivity or importance. The lines are thin and gray, creating a subtle background pattern.

Resultados

Influência do tamanho do problema

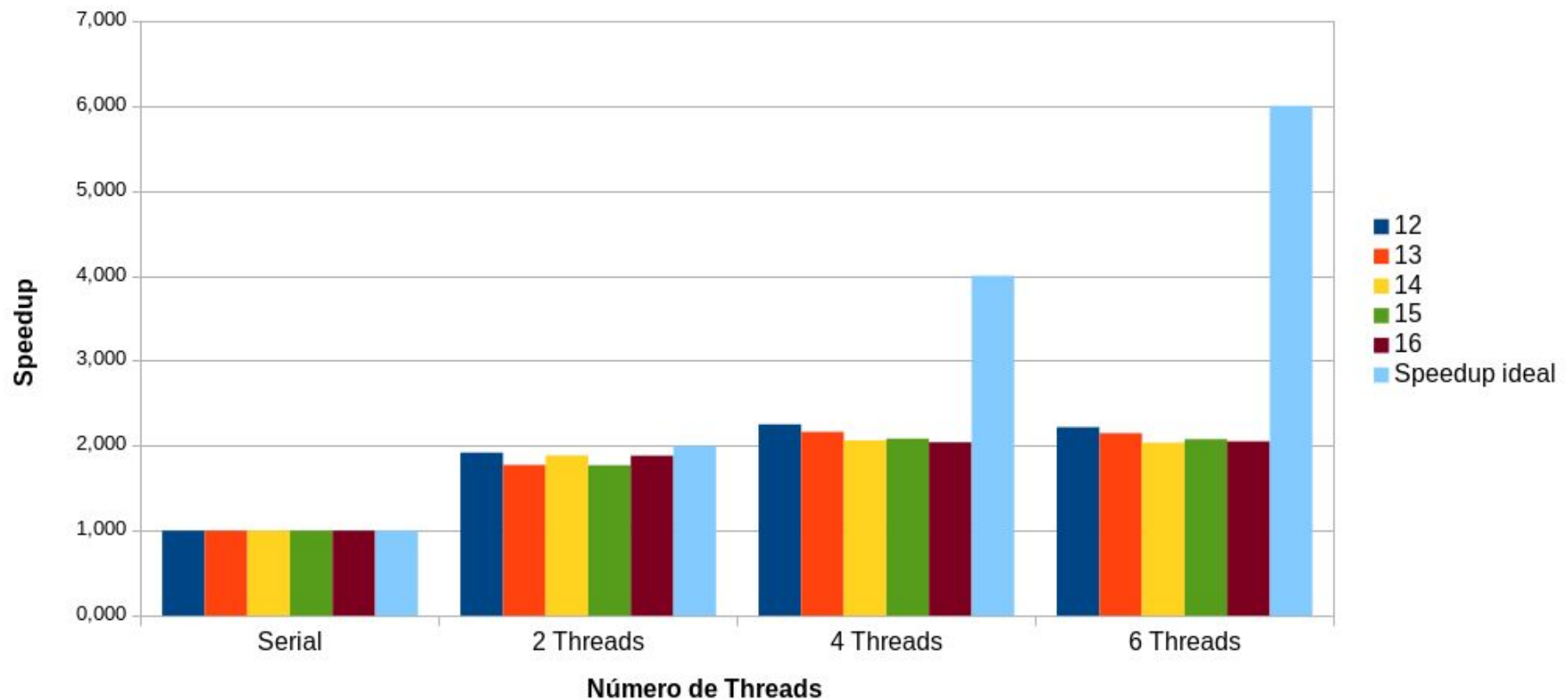
Relação speedup com tamanho do problema (N)



Influência do tamanho do problema

Relação speedup com o tamanho do problema (N)

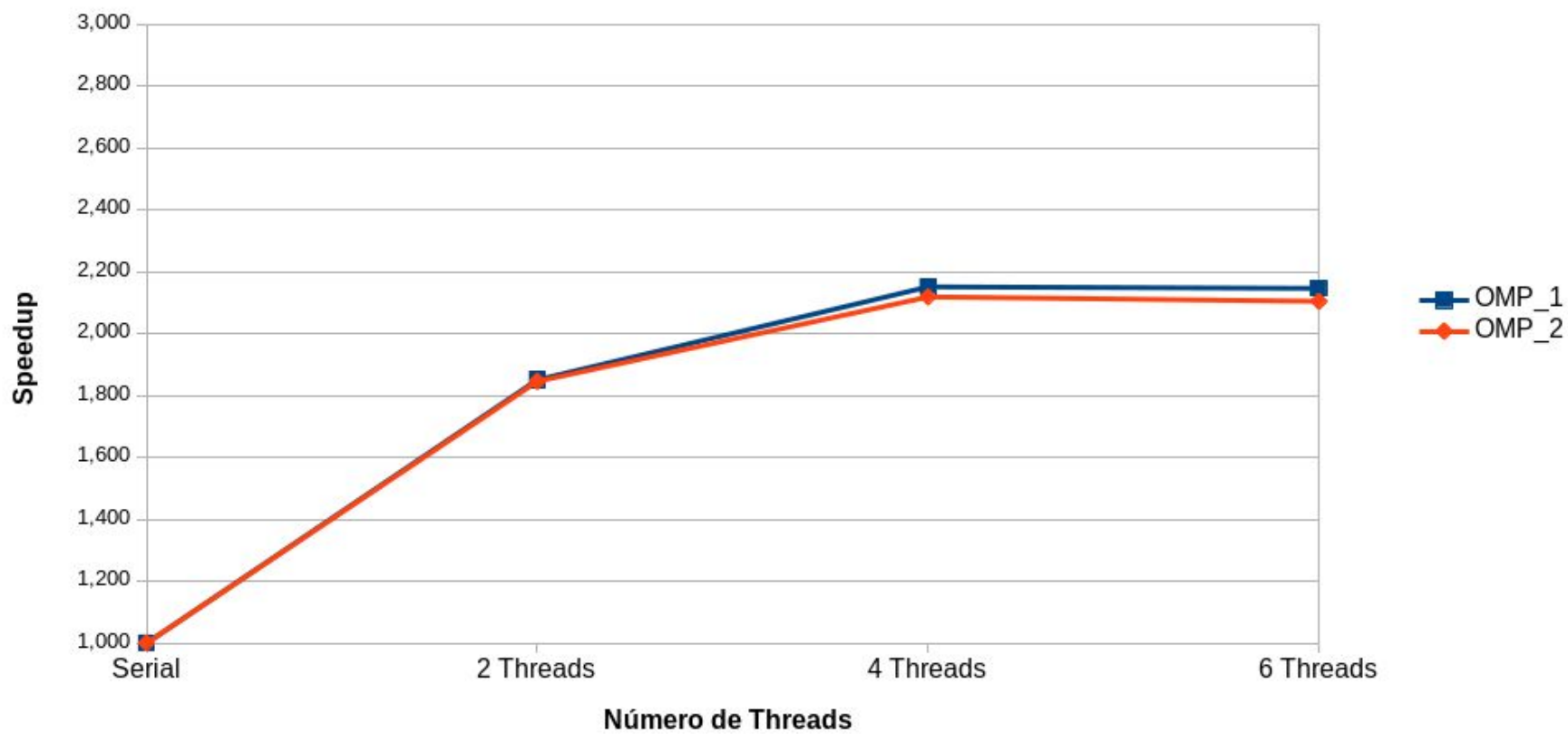
Implementação 2



Comparação das versões implementadas com OpenMP

Comparação entre implementações

Variação de schedule



Conclusões

- ◎ **Implementação 1** mostrou-se ligeiramente **melhor** que a implementação 2 no caso geral:
 - Melhor distribuição da carga de trabalho
- ◎ Mesmo que as **versões paralelizadas** obtenham melhor desempenho, o speedup tende a se **estagnar dependendo do número de threads**;
- ◎ Quanto **maior o número de soluções**, **menor o speedup**;
- ◎ Tentativa de utilização de outras estratégias de paralelização...

```
void nqueens(int size, int *solutions) {
    int i, count;
    int* position;

    count = 0;

    for(i=0; i<size; i++) {
        int j;
        position = (int *) malloc(size * sizeof(int));
        position[0] = i;

        for(j = 1; j < size; j++)
            position[j] = -1;

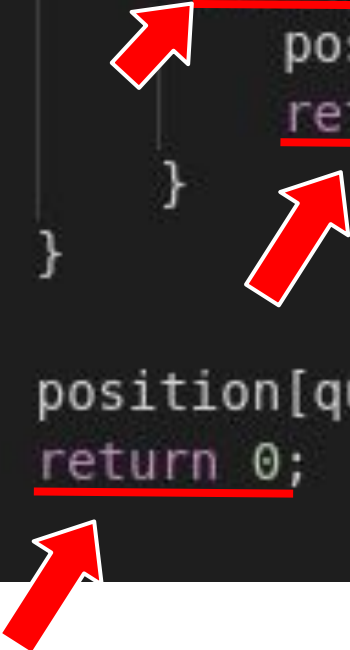
        int queen_number = 1;
        while(queen_number > 0) {
            if(put_queen(size, queen_number, position)) {
                queen_number++;
                if(queen_number == size) {
                    count += 1;

                    position[queen_number-1] = -1;
                    queen_number -= 2;
                }
            } else {
                queen_number--;
            }
        }
    }

    *solutions = count;
}
```

Conclusões

```
int put_queen(int size, int queen_number, int* position) {  
    int i;  
    for(i=position[queen_number]+1; i<size; i++) {  
        if(ok(queen_number, i, position)) {  
            position[queen_number] = i;  
            return 1;  
        }  
    }  
  
    position[queen_number] = -1;  
    return 0;  
}
```



Conclusões

```
int ok(int queen_number, int row_position, int* position) {  
    int i;  
  
    // Check each queen before this one  
    for(i = 0; i < queen_number; i++) {  
        // Get another queen's row_position  
        int other_row_pos = position[i];  
  
        // Now check if they're in the same row or diagonals  
        if (other_row_pos == row_position || // Same row  
            other_row_pos == row_position - (queen_number - i) || // Same diagonal  
            other_row_pos == row_position + (queen_number - i)) // Same diagonal  
            return 0;  
    }  
    return 1;  
}
```

