



Parallel Mandelbrot Set

Lucas Ferreira da Silva

Carga de trabalho

- ◎ Concentrada no **laço principal** da função *main*;
- ◎ 3 Laços **aninhados**;
- ◎ **Desbalanço de carga** entre os laços **for** e **while**.

```

int main(){
    int max_row, max_column, max_n;
    cin >> max_row;
    cin >> max_column;
    cin >> max_n;

    char **mat = (char**)malloc(sizeof(char*)*max_row);

    for (int i=0; i<max_row;i++)
        mat[i]=(char*)malloc(sizeof(char)*max_column);

    for(int r = 0; r < max_row; ++r){
        for(int c = 0; c < max_column; ++c){
            complex<float> z;
            int n = 0;
            while(abs(z) < 2 && ++n < max_n)
                z = pow(z, 2) + decltype(z)(
                    (float)c * 2 / max_column - 1.5,
                    (float)r * 2 / max_row - 1
                );
            mat[r][c]=(n == max_n ? '#' : '.');
        }
    }

    for(int r = 0; r < max_row; ++r){
        for(int c = 0; c < max_column; ++c)
            std::cout << mat[r][c];
        cout << '\n';
    }
}

```


**Concentração
de trabalho**

A decorative network diagram in the top-left corner, featuring a complex web of interconnected nodes and lines. The nodes are represented by circles of varying sizes, some with concentric rings, and the lines are thin and grey. The diagram is partially cut off by the left edge of the slide.

1ª Implementação

Implementação 1

- ◎ **OpenMP;**
- ◎ Paralelização do **laço mais externo;**
- ◎ Definição do *schedule* ***dynamic;***




```
int main(){
    int max_row, max_column, max_n;
    cin >> max_row;
    cin >> max_column;
    cin >> max_n;

    char **mat = (char**)malloc(sizeof(char*)*max_row);

    for (int i=0; i<max_row;i++)
        mat[i]=(char*)malloc(sizeof(char)*max_column);

    #pragma omp parallel for schedule(dynamic)
    for(int r = 0; r < max_row; ++r){
        for(int c = 0; c < max_column; ++c){
            complex<float> z;
            int n = 0;
            while(abs(z) < 2 && ++n < max_n)
                z = pow(z, 2) + decltype(z)(
                    (float)c * 2 / max_column - 1.5,
                    (float)r * 2 / max_row - 1
                );
            mat[r][c]=(n == max_n ? '#' : '.');
        }
    }

    for(int r = 0; r < max_row; ++r){
        for(int c = 0; c < max_column; ++c)
            std::cout << mat[r][c];
        cout << '\n';
    }
}
```






2ª Implementação



Implementação 2

- ◎ **OpenMP;**
- ◎ Paralelização do **laço mais externo;**
- ◎ Definição do *schedule dynamic*;
- ◎ Otimização da **escrita do resultado.**





```
int main(){
    int max_row, max_column, max_n;
    cin >> max_row;
    cin >> max_column;
    cin >> max_n;

    char **mat = (char**)malloc(sizeof(char*)*max_row);

    for (int i=0; i<max_row;i++)
        mat[i]=(char*)malloc(sizeof(char)*max_column);

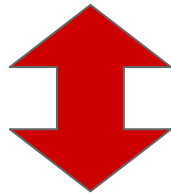
    #pragma omp parallel for schedule(dynamic)
    for(int r = 0; r < max_row; ++r){
        for(int c = 0; c < max_column; ++c){
            complex<float> z;
            int n = 0;
            while(abs(z) < 2 && ++n < max_n)
                z = pow(z, 2) + decltype(z)(
                    (float)c * 2 / max_column - 1.5,
                    (float)r * 2 / max_row - 1
                );
            mat[r][c]=(n == max_n ? '#' : '.');
        }
    }

    for(int r = 0; r < max_row; ++r){
        cout << string(mat[r]) << endl;
    }
}
```



“Buffer” para output

```
for(int r = 0; r < max_row; ++r){  
    for(int c = 0; c < max_column; ++c)  
        std::cout << mat[r][c];  
    cout << '\n';  
}
```



```
for(int r = 0; r < max_row; ++r){  
    cout << string(mat[r]) << endl;  
}
```

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 are thin and gray, connecting the nodes in a non-linear fashion.

Experimentos

Experimentos

- ◎ 10 execuções de cada configuração;
- ◎ Execução das 3 versões:
 - **Serial**
 - **OpenMP**
 - **OpenMP-buffer**
- ◎ Entrada:
 - 115 > Número máx. de linhas
 - 395 > Número máx. de colunas
 - 12000 > Número de iterações

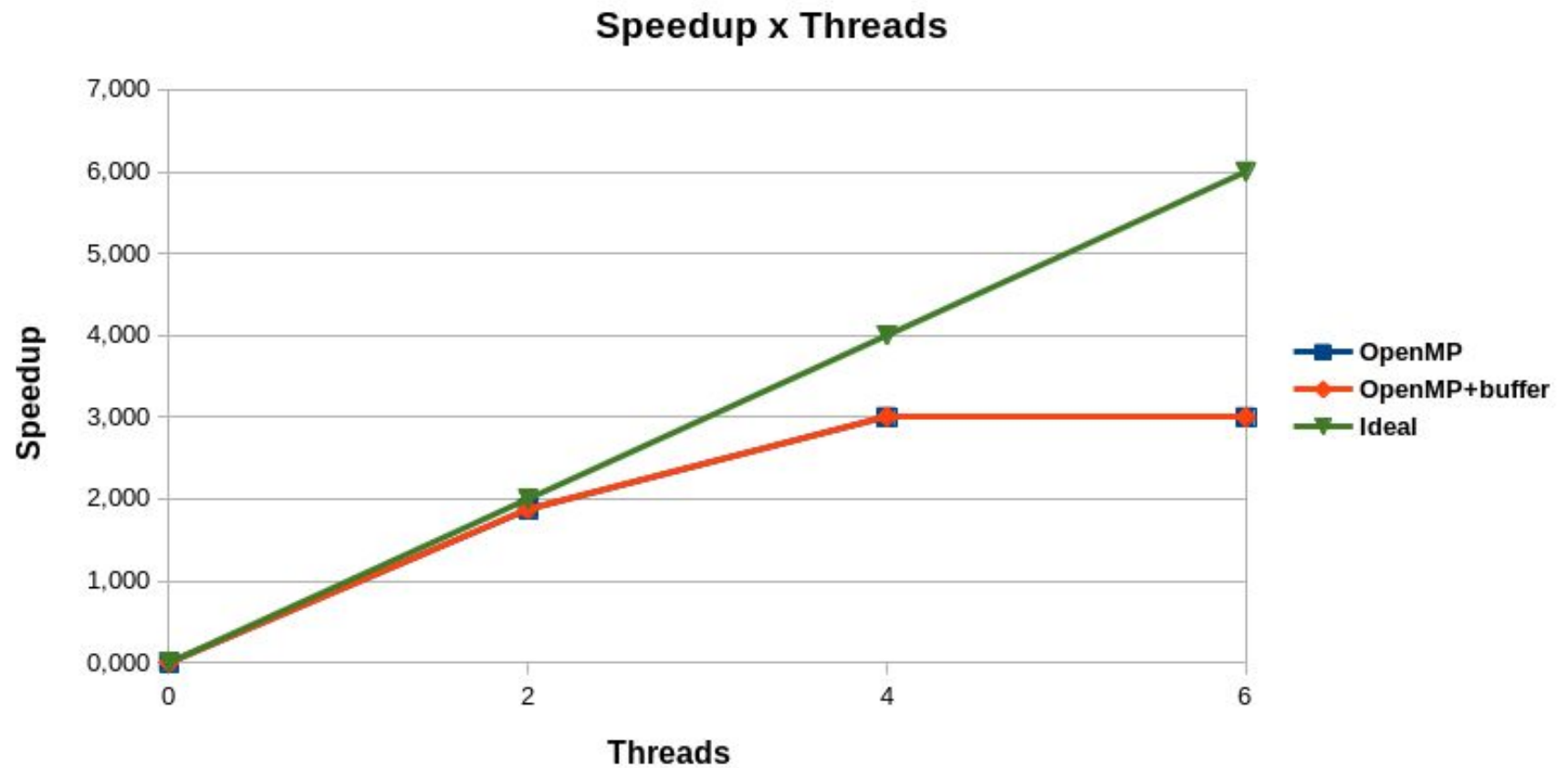
Experimentos

- ◎ Variação entre **2**, **4** e **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 a hierarchical or multi-layered structure. The lines are thin and gray, connecting the nodes in a non-linear fashion.

Resultados

Speedup

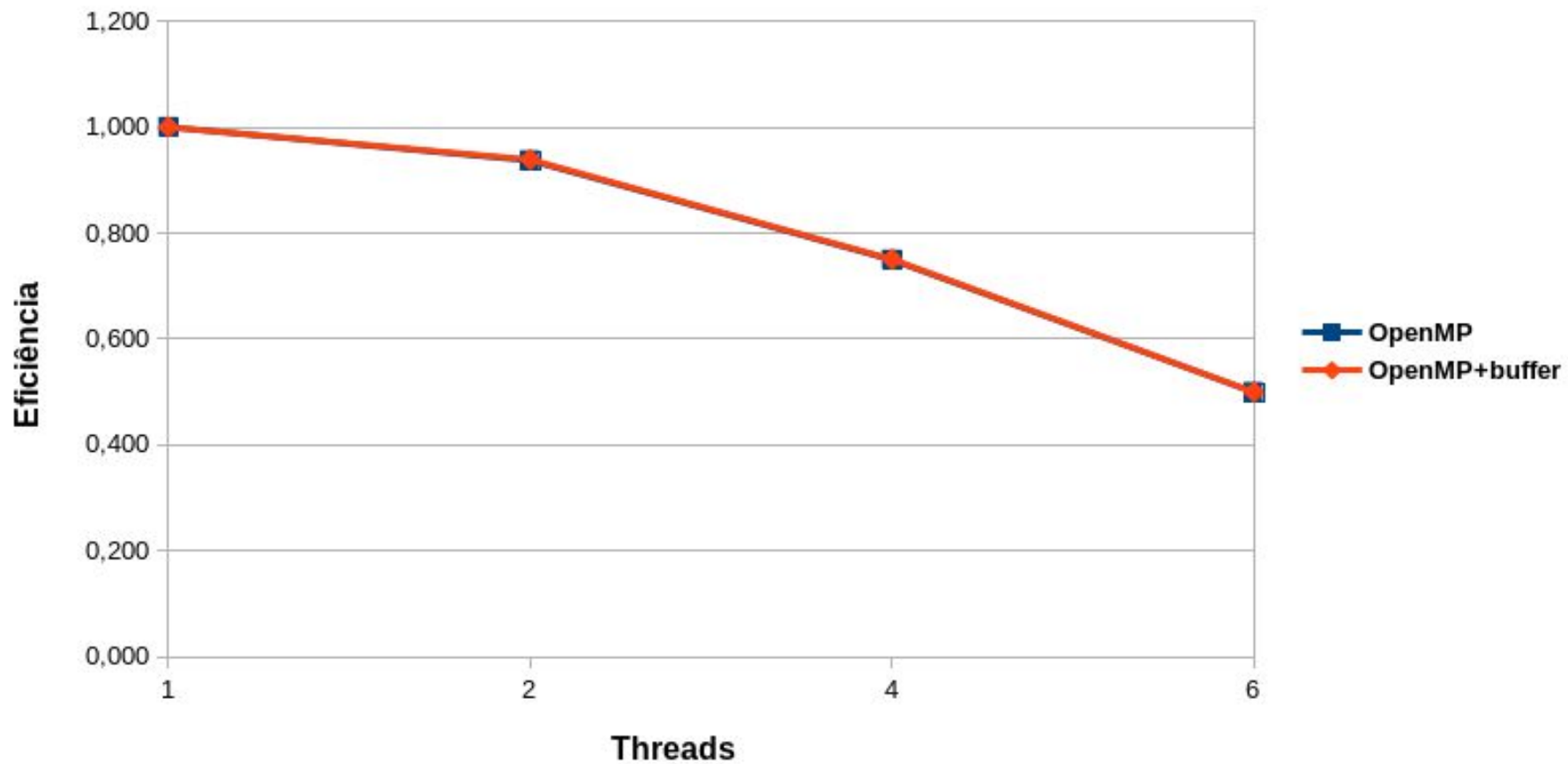


Speedup

Threads	OpenMP	OpenMP+Buffer	Ideal
2	1,874	1,877	2,000
4	2,998	3,002	4,000
6	2,995	2,997	6,000

Eficiência

Eficiência x Threads



Eficiência

Threads	OpenMP	OpenMP+Buffer
2	0,937	0,939
4	0,749	0,751
6	0,499	0,500

Conclusões

- ◎ **Solução** bastante **simples** com OpenMP.
- ◎ **Ambas** as implementações com **OpenMP** obtiveram bom speedup e boa eficiência.
- ◎ A **segunda** implementação com **OpenMP** se mostrou **minimamente melhor** (quase nada) que a primeira implementação.
 - Talvez ajudasse em pontos para critério de desempate em uma maratona...