UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMAÇÃO DISTRIBUÍDA E PARALELA (INF01008)

Material Estendido - Colorize

Nome: Henrique Goetz **Matrícula**: 00274719

1) Introdução

Durante a apresentação dos resultados do algoritmo, algumas dúvidas surgiram e esse material tem como objetivo esclarecer e verificar os resultados sobre os pontos levantados.

Esse conteúdo é uma extensão dos resultados apresentados no pdf Colorize.pdf que foi entregue junto deste relatório. O material completo está disponível em: https://github.com/HenriqueGoetz/colorize-c.

2) Análise do tempo de execução do trecho escolhido para paralelização

Para acelerar o tempo de execução, se optou por realizar a paralelização de um loop "*for*" utilizando "*#pragma omp parallel for*". Essa escolha ocorreu por se acreditar que esse era o ponto de maior complexidade do programa. Entretanto, não se tinha realizado nenhuma medição para comprovar esse gargalo.

Então para sanar essa dúvida, se realizou uma execução do programa sequencial registrando o tempo total de execução e o tempo gasto no "*for*" citado. Segue o resultado:

```
henrique@henrique:~/workspace/colorize-c$ ./colorize gs/image-1.png cf/image-1.png op/image-1-new.png
Tempo de Colorize: 108.974482
Tempo de Execução Total: 109.025701
henrique@henrique:~/workspace/colorize-c$
```

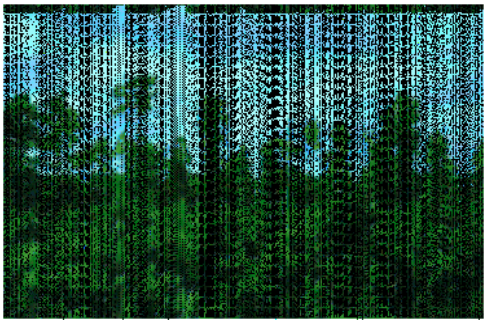
Como pode se perceber, a escolha estava correta. O tempo de execução do trecho escolhido para a paralelização corresponde a quase todo o tempo do programa. Abaixo um novo resultado agora utilizando o programa concorrente:

```
henrique@henrique:~/workspace/colorize-c$ ./colorize gs/image-1.png cf/image-1.png op/image-1-new.png
Tempo de Colorize: 38.333241
Tempo de Execução Total: 38.411258
henrique@henrique:~/workspace/colorize-c$
```

3) Metodologia para chegar no código final

Após escolher o trecho que deveria executar de forma concorrente, foi necessário adaptar o código para conseguir uma execução consistente. Inicialmente, o código não gerou o resultado esperado. Abaixo segue como o código estava implementado e o resultado:

```
void colorize_image() {

    int i, j;

    #pragma omp parallel for
    for(i = 1; i < dest_width - 1; i++) {
        for(j = 1; j < dest_height - 1; j++) {


            /* ... */
```



Após analisar o código novamente, percebeu-se que a variável *"j"*, declarada fora do loop, estava sendo acessada sem sincronização, o que corrompeu seu valor durante a execução.

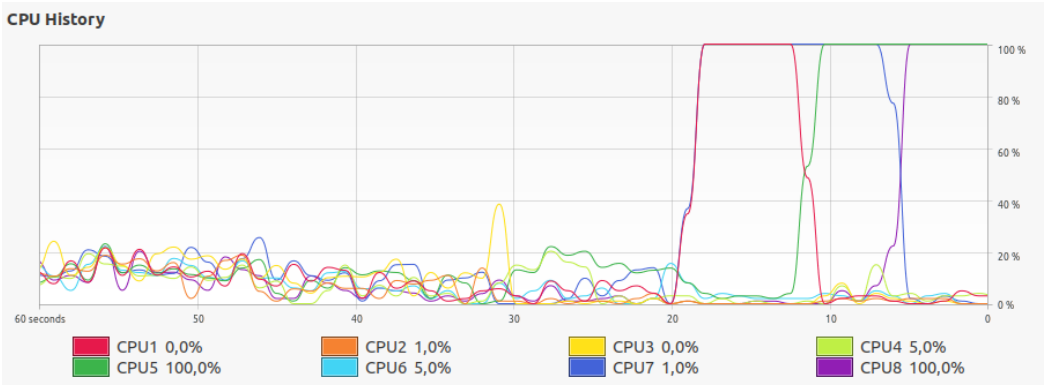
Para resolver esse problema, se optou por declarar a variável dentro do loop, assim a variável torna- se privada para cada execução do *"for"*. Segue o trecho corrigido e o resultado:

```
void colorize_image() {

    int i;

    #pragma omp parallel for
    for(i = 1; i < dest_width - 1; i++) {
        int j;
        for(j = 1; j < dest_height - 1; j++) {


            /* ... */
```

4) Variação no número de Threads

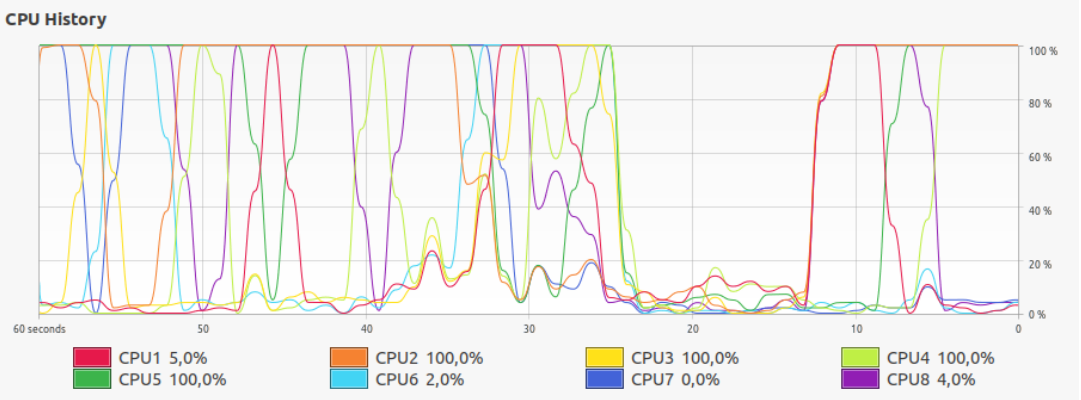
Na apresentação se comparou o desempenho dos algoritmos e, na versão concorrente o número de threads era 8. Para verificar o impacto desse parâmetro, realizou mais uma coleta de resultados.

4.1) Número de Threads: 2



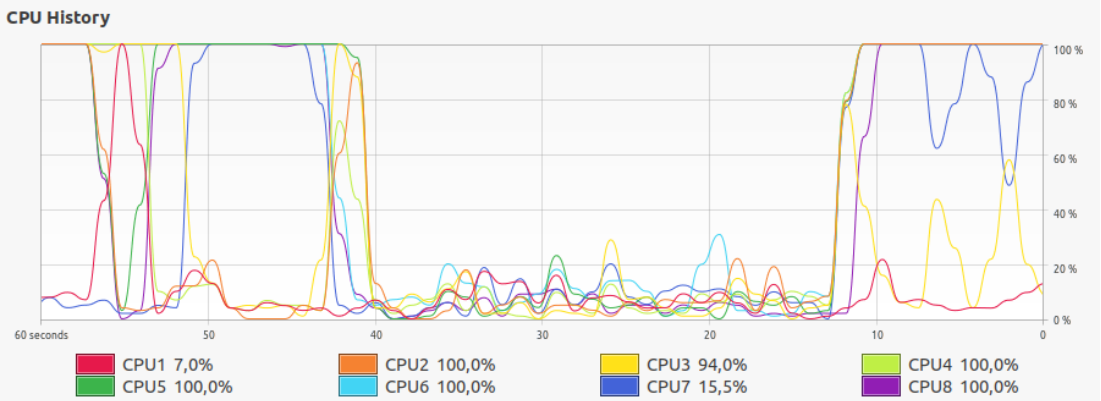
Tempo de Execução: 64s

4.2) Número de Threads: 4



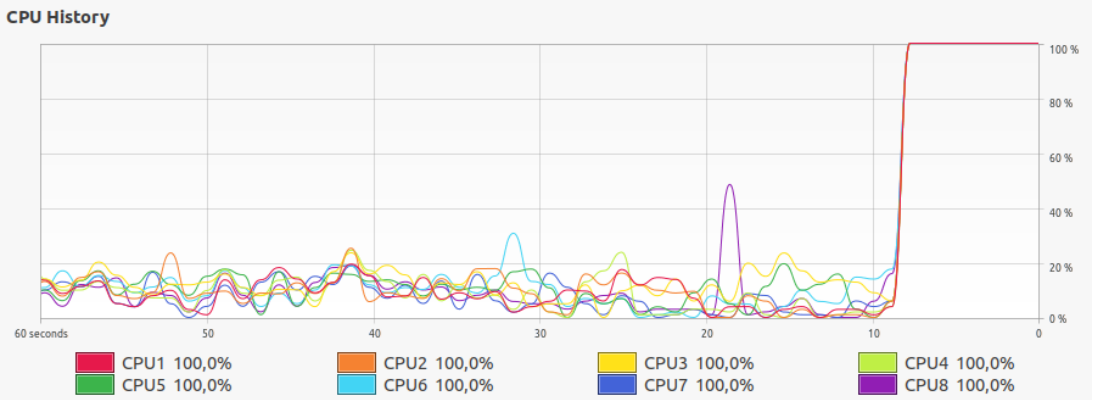
Tempo de Execução: 41s

4.3) Número de Threads: 6

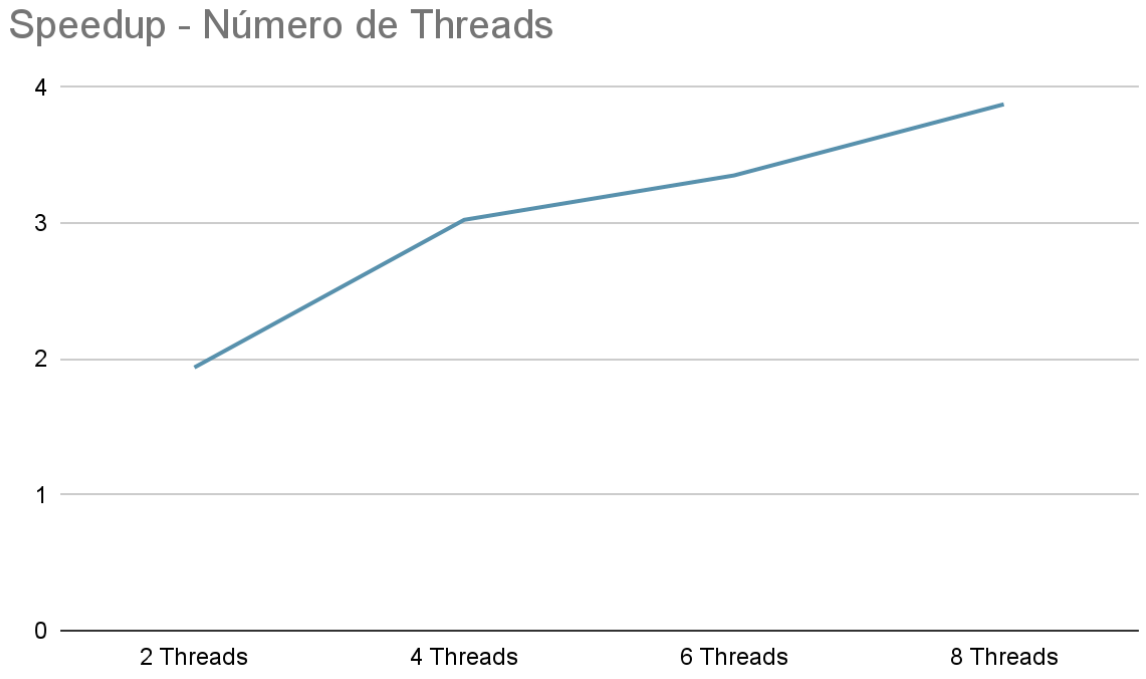


Tempo de Execução: 37s

4.3) Número de Threads: 8



Tempo de Execução: 32s


Speedup - Número de Threads
4
3
2
1
0
2 Threads
4 Threads
6 Threads
8 Threads